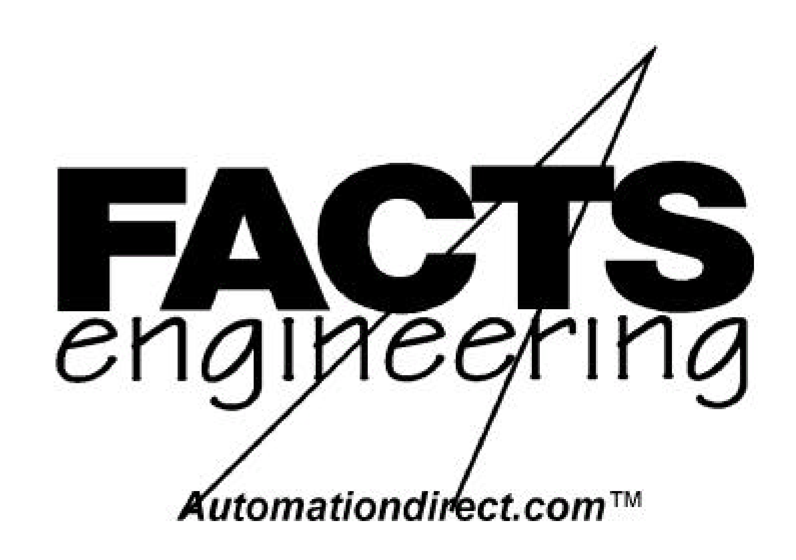


MANUAL DE REFERENCIA DE BASIC DE FACTS ENG

No. de artículo: F0-CP-M-SP

MARCAS REGISTRADAS

PROPIEDAD INTELECTUAL (COPYRIGHT)

ADVERTENCIA

Gracias por comprar el equipo de automatización de FACTS ENGINEERING. Deseamos que su nuevo equipo de automatización de FACTS ENGINEERING funcione con seguridad. Cualquier persona que instala o aplica este equipo debe leer esta publicación (y cualquiera otra publicación relevante) antes de instalar o de hacer funcionar el equipo.

Para reducir al mínimo el riesgo potencial de problemas de seguridad, usted debe seguir todos los códigos locales y nacionales aplicables que regulen la instalación y la operación de su equipo. Estos códigos varían de área a área y cambian generalmente con el tiempo. Es su responsabilidad determinar qué códigos deben ser seguidos, y verificar que el equipo, la instalación, y la operación esté de acuerdo con la última revisión de estos códigos.

Como mínimo, usted debe seguir todas las secciones aplicables del National Fire Code, National Electrical Code, y los reglamentos de National Electrical Manufacturers Association (NEMA) de Estados Unidos. Otras oficinas gubernamentales reguladoras o locales pueden ayudar a determinar qué códigos y estándares son necesarios para instalación y operación seguras.

Pueden resultar daños al equipo o lesión seria al personal al no seguir todos los reglamentos y estándares aplicables. No garantizamos que los productos descritos en esta publicación son convenientes para su aplicación particular, ni asumimos cualquier responsabilidad del diseño de su producto, instalación, u operación.

Si usted tiene cualquier pregunta referentes la instalación o a la operación de este equipo, o si usted necesita la información adicional, llámenos por favor en 1-800-783-3225.

Este documento se basa en la información disponible a el momento de su publicación. Aunque se han hecho esfuerzos de ser precisos, la información contenida no pretende cubrir todos los detalles o variaciones en hardware y software, ni prever cada contingencia posible en la conexión con la instalación, la operación, y el mantenimiento. Pueden ser descritas características que no están presentes en todos los sistemas de hardware y de software. FACTS ENGINEERING no asume ninguna obligación de aviso a los lectores de este documento con respecto a cambios realizados posteriormente.

FACTS ENGINEERING conserva el derecho de realizar cambios al hardware y al software en cualquier momento, sin aviso previo.

FACTS ENGINEERING no hace ninguna representación o garantía, expresada, implicada, o estatutaria con respecto a, y no asume ninguna responsabilidad de la exactitud, de lo completo, de la suficiencia o de la utilidad de la información contenida adjunto. No se aplicará ninguna garantía de mercadologia de aptitud para el propósito.

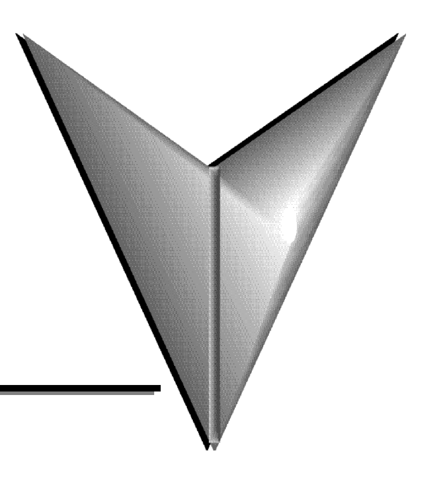
HISTORIA DE REVISIONES DEL MANUAL DE REFERENCIA BASIC DE FACTS ENG.

Por favor incluya el número del manual y la edición del manual, ambos mostrados abajo, cuando se comunique con Apoyo Técnico en relación a este documento.

Número del Manual : FA-BASIC-M-SP
Edición: Primera edición en español
Fecha de edición: 12/05

Historia de Publicaciones		
Edición	**Fecha**	**Descripción de revisiones**
Original	12/99	Original issue in English
Primera Edicion	1/06	Traducción por el Ing. Luis Miranda, miembro del equipo de Apoyo Técnico de Automationdirect.

CONTENIDO



Capítulo 1: Introducción

Capítulo 2: Como comenzar con BASIC ampliado de Facts Eng.

CAPÍTULO 3: COMANDOS DEL SISTEMA

Capítulo 4: Declaraciones

Capítulo 5: Operadores matemáticos

Capítulo 6: Operadores lógicos y de comparación

Capítulo 7: Mensajes de error

Capítulo 8: Avanzado

Apéndice A: Como finalizar un programa

Apéndice B: Palabras reservadas

Apéndice C: Tabla ASCII

Apéndice D: Velocidad de ejecución de programas BASIC

Apéndice E: Lista de declaraciones y de operadores

INTRODUCCIÓN



En este capítulo...

1 Introducción

Propósito de este documento

Este documento es una traducción del documento en inglés del mismo nombre y describe el compilador BASIC ampliado de FACTS Eng. que se usa en los módulos de productos que usan BASIC de FACTS Eng..

Se pretende que este documento sea usado conjuntamente con el manual de usuario específico del módulo que se ha comprado. Este manual describe los comandos, las declaraciones, y la información de carácter general sobre el compilador. La información que es específica a un módulo particular tal como especificaciones del módulo, clavijas de los puertos e instrucciones específicas del módulo se documentan en el manual de usuario de ese módulo.

Este manual contiene numerosos ejemplos de uso y de programación, sin embargo, se asume que el usuario tiene cierto conocimiento de BASIC u otra experiencia de programación de un lenguaje de un nivel más alto. Esto no es un manual de "cómo escribir el programa de lógica BASIC/Ladder".

Quién debe leer este manual

Este manual contiene información importante para los que instalen, mantengan, y/o hagan funcionar qualquiera de los modulos Coprocessor CP128.

Apoyo Técnico

Apoyo de FACTS Eng.

Por Teléfono: 1-800-783-3225 en Estados Unidos o
Por fax 727-375-5441
(Lunes a Viernes, 9:00 a.m.-6:00 p.m. E.T.)
En Internet: www.facts-eng.com

Apoyo de AutomationDirect

Por Teléfono: 770-844-4200
Por fax 770-886-3199
(Lunes a Viernes, 9:00 a.m.-6:00 p.m. E.T.)
En Internet: www.automationdirect.com

Nuestros grupos de apoyo técnico trabajarán con usted para contestar sus preguntas.

Si no puede encontrar la solución para su aplicación, o si por cualquier otra razón usted necesita ayuda técnica adicional, por favor llame a Apoyo Técnico. Estamos disponibles los días de semana de 9:00 a.m. hasta las 6:00 p.m. Hora del Este de Estados Unidos.

Además le invitamos a que visite nuestro sitio en Internet, donde puede encontrar información técnica y no técnica sobre nuestros productos y nuestras empresas.

Convenciones usadas

argumentos Los argumentos de una instrucción o declaración son mostrados en itálico

CTRL-C El tecleado de un conjunto de teclas será indicado de esta forma.

RENUMBER Una instrucción o declaración que se use en el programa es mostrada en mayúsculas

```
1085 PRINT1 $(I) “=”
1090 NEXT :REM ejemplo de programa
PRM0
READY
```

El programa es mostrado en negrilla **arial,** 9 puntos**.**

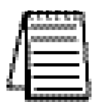
Usamos las siguientes palabras como traducción de la palabra en inglés:

Save: Archivar: Quiere decir guardar el programa en un disco o sistema de almacenaje

String: Cadena de caracteres.

Download: Bajar (copiar un programa desde la PC a la memoria del módulo)

En el texto original se usa alternativamente **ABM** para denotar genéricamente el módulo Coprocessor.



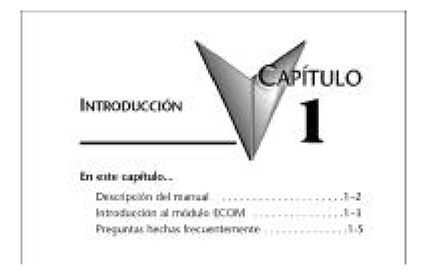
NOTA: *Cuando vea el icono de la "libreta" en el margen de la izquierda, el párrafo en el lado derecho será una nota especial..*
*La palabra **NOTA:** en negillla marcará el inicio del texto.*



Cuando vea el icono del "punto de exclamación" en el margen de la izquierda, el párrafo a la derecha será uno de ADVERTENCIA. Esta información puede evitar heridas, pérdidas de propiedad, o (en casos extremos) hasta la muerte.
La palabra ADVERTENCIA: en negrilla marcará el inicio del texto.

Asuntos claves de cada capítulo

El inicio de cada capítulo hará una lista de los tópicos principales encontrados en ese capítulo.



COMO COMENZAR CON BASIC AMPLIADO DE FACTS ENG.



En este capítulo...

Requisitos mínimos de lectura

2

Los usuarios que usan por la primera vez este programa ya familiares con la programación de BASIC deben repasar por lo menos los comandos AUTOSTART, NEW, LIST, SAVE y DELPRM. También repasar las declaraciones SETPORT, SETINPUT y el manual de usuario específico al módulo que será utilizado.

Usuarios primerizos

Se recomienda que los usuarios que usan por la primera vez este programa comiencen entrando y ejecutando los ejemplos en la sección de "COMIENZO RÁPIDO" en el apéndice A del manual de usuario específico del módulo. Esta sección guía al usuario con los diversos pasos del desarrollo de un programa en BASIC.

El BASIC ampliado de FACTS Eng. se basa en el compilador del MCS Basic-52 con muchos mejoramientos e instrucciones agregadas orientadas para control. El BASIC ampliado de FACTS Eng. lee, interpreta, y ejecuta una lista de instrucciones que se almacenan en la memoria del módulo. Esta lista de instrucciones es el programa de usuario. El programa es escrito y cargado en la memoria por el usuario. La funcionalidad del programa es determinada por las instrucciones contenidas en el programa.

Modos de funcionamiento

El intérprete del BASIC ampliado de FACTS Eng. funciona en dos modos: El modo directo o COMMAND o el modo RUN (operación).

Los comandos pueden ser entrados solamente cuando el módulo está en el modo COMMAND. El intérprete de BASIC toma acción inmediata después de que se haya entrado un comando.

Se entra, corrige, modifica, lista y copia programas en el modo COMMAND. El módulo se puede programar para entrar en cualquier modo después de un reset o al energizar el módulo con el comando de AUTOSTART.

Reset

Un reset del módulo ocurre bajo las siguientes condiciones para los tipos siguientes de módulos:

módulos 305 de BASIC

- ocurre una energización
- el usuario teclea el comando RESET en el aviso de comando
- la CPU 305 va al modo de funcionamiento

Nota: El módulo BASIC de memoria de 64K del usuario 305 no puede ser alcanzado hasta que la CPU 305 está en modo RUN. Los módulos 305 de memoria de usuario de 128K se pueden seleccionar por colocación de puentes para hacer reset cuando la CPU 305 va al modo de programa.

Los módulos 305 de la CPU BASIC, los módulos 405 de CoProcessor y los módulos

- el usuario teclea el comando RESET en el aviso de comando

En casos raros, se puede también generar un reset por el temporizador a bordo del watchdog.

Cuando ocurre un reset el intérprete verifica el modo corriente de AUTOSTART. El modo AUTOSTART determina lo que hará el intérprete. Vea AUTOSTART para una descripción detallada. De acuerdo con los parámetros corrientes en AUTOSTART, el intérprete esperará un carácter de barra de espacio en el puerto de comando, hará funcionar un programa especificado, o imprimirá un mensaje de energización desde del puerto de comando y dará un aviso de comando.

Uso general de memoria

Todos los módulos BASIC de FACTS Eng. (305 BASIC, BASIC de 305 CPU, 405 CoProcessor, 205 CoProcessor) tienen la misma disposición general de memoria del programa. Típicamente, en los programas se eliminan errores (DEBUG) en memoria de datos y después se guardan y operan desde la memoria de programa. Los programas se pueden también salvar al disco duro usando el programa de configuración y software de documentación incluído **ABM Commander Plus** en una PC compatible de IBM.

La memoria de datos y la memoria del programa son ambas respaldadas por batería. Se usam dos baterías encapsuladas de litio (contenidas en el soquete de RAM) para mantener la memoria RAM. Estas baterías no son reemplazables y se puede esperar mantener datos y programas en RAM por más de 10 años.

Memoria de datos

La memoria de datos es el segmento de memoria que se utiliza para la edición de programas y desarrollo. Todos los programas almacenan variables en esta memoria. Esta memoria también se refiere como banco 0 o PROGRAMA 0.

El lenguaje BASIC ampliado de FACTS Eng. orientado a un ambiente de control está contenido en 32K o 64K bytes de ROM no direccionables (vea el manual de usuario específico del módulo). Una porción de la memoria de datos es reservada para el uso del intérprete BASIC. La cantidad de memoria reservada depende del tipo específico del módulo.

El PROGRAMA 0 es el programa almacenado en la memoria de datos. Puede ser ejecutado automáticamente por el comando AUTOSTART o por otro programa (por ejemplo, 1000 GO_PROGRAM 0).

El PROGRAMA 0 puede ser una copia protegida por medio de la declaración LOCKOUT.

Memoria del programa

Todos los módulos suministran al usuario una porción de memoria designada memoria de programa. Se usa este segmento de memoria para guardar o para archivar programas (SAVE). Los programas archivados en un archivo de memoria de programa se pueden mover de nuevo a la memoria de datos (véase el comando EDIT) para ejecutar modificaciones, eliminar errores, para tratar de ejecutar pruebas y después archivar nuevemente en el archivo de la memoria de programa.

Se pueden archivar múltiples programas en la memoria de programa para crear un archivo del aplicación y de programas utilitarios.

Los programas se pueden ejecutar directamente desde la memoria de programa por el comando AUTOSTART o por otros programas con la declaración de GO_PROGRAM. Los programas se pueden también “ENCADENAR" juntos con el modo 2 de AUTOSTART.

Definición de los comandos

Comandos

1. El carácter de aviso ">"es enviado por BASIC y mostrado en la pantalla donde está instalado ABM Commander Plus, para informar al usuario que está en el modo COMMAND y para dejarlo listo para recibir caracteres.

2– Los comandos pueden ser entrados solamente cuando el módulo está en el modo COMMAND.

3. BASIC toma acción inmediata después de que se haya entrado un comando.

4. Los comandos que comienzan con un número a partir de 0 a 65535 se interpretan como líneas de programa y se terminan con un carácter de **retorno de carro**.

5. Muchas de las instrucciones y todos los operadores se pueden entrar sin el número de línea y pueden ser ejecutadas inmediatamente. Esto una herramienta de "debugging" (eliminación de errores) de gran importancia.

```
>PRINT1 21*196.3
4122–3

>FOR I=0 TO 12 : P. 2**I, " ", : NEXT I
1 2 4 8 16 32 64 128 256 512 1024 2048 4096

>B=10
>CONT

>PH0. 97
61H
>$(0)="?"
>P. ASC($(0),1)
>63
```

6. Los comandos que no se pueden incluir en líneas de programa serán presentados en el CAPÍTULO 3: COMANDOS DE SISTEMA. Algunos comandos típicos del sistema son RUN, LIST, SAVE y NEW.

Declaraciones

Una declaración consiste en una instrucción (Por ejemplo, PRINT, INPU, LET, GOTO) y puede incluir números, variables, operadores y números de línea. Los programas de aplicaciones se construyen con declaraciones.

Líneas de programa

1. Cada línea de programa contiene una declaración. Se pueden entrar declaraciones múltiples en una sola línea si son separadas por dos puntos (:).

2– La ejecución de las líneas de programa ocurre cuando se manda al módulo a hacer funcionar un programa. Vea AUTOSTART, RUN, GOTO, GO_PROGRAM.

3. Una línea de programa no puede contener más de 79 caracteres.

4. Las líneas de programa no necesitan ser entradas en orden numérica, porque BASIC usará los números de línea para pedir ordenar el programa secuencialmente.
5. Un número de línea de programa se puede usar solamente una vez en un programa y solamente se permite un número de línea en cada línea de programa.



NOTA: Si se entra el mismo número de línea múltiples veces, entonces la última línea introducida sobreescribirá a la anterior.

6. Los espacios (espacios en blanco) entrados en líneas de programa entre instrucciones, operadores, variables, expresiones y números son ignoradas por BASIC; sin embargo, BASIC insiere automáticamente espacios durante un LISTado para mejorar el aspecto y la legibilidad del programa.
7. Las líneas de programa comienzan con un número en el rango de 0 a 65535 y se terminan con un **retorno de carro**.

Números de coma flotante

1. Los números de coma flotante tienen un rango de ±E-127 a ±.99999999E+127.

2– Los números de coma flotante pueden entrar y salir usando dos notaciones diferentes.

A. Coma flotante fraccionaria (93,65)

B. Coma flotante exponencial (39,6537E+6)

El BASIC redondea números de coma flotante a ocho dígitos significativos.

4. Cada número de coma flotante requiere seis bytes de memoria para el almacenaje.

Números enteros

1. Los números enteros tienen un rango de 0 a 65535 (0FFFFH).

2– Los números enteros pueden entrar y salir usando dos notaciones diferentes.

A. Número entero Decimal (127)

B. Número entero Hexadecimal (0A53H)

3. Los números enteros que se representan en formato hexadecimal deben comenzar con un dígito válido de modo que puedan ser distinguidos de variables (A0H es entrado 0A0H).
4. Cuando los operadores lógicos de BASIC, tales como .OR. requieren un número entero, BASIC truncará la parte fraccionaria del número dejando la porción del número entero para la operación.
5. Los números enteros requieren seis bytes de memoria para el almacenaje.

Operadores

1. Los operadores realizan una función predefinida. Los operadores tales como RND y LOF vuelven un número y no requieren un argumento. Los operadores tales como SIN y ABS requieren un argumento en el cual se realice la operación. Algunos operadores que requieren dos argumentos para realizar la operación son AND, + (sumar), - (restar), e = (igual).

2– Los operadores se distinguen por su tipo. Los dos tipos generales de operadores son:

A. Matemáticos

B. Lógicos y de comparación

Variables

1. Una variable debe comenzar con una letra y puede contener hasta 79 caracteres o números incluyendo el carácter sublineado (_). Las variables válidas son:

   S0L_1 RELAY10 INP103 REG410

TMRACCUMM1 = TMRPRESET1 puesto que ambas variables tienen la misma longitud, ambas comienzan con "T" y ambas terminan con "1".



***NOTA**: Solamente la cantidad de caracteres en el nombre de la variable y de los primeros y últimos caracteres es significativa para identificar únicamente a la variable.*

En vez de usar nombres de variables tales como REG0, REG1, REG10, y REG20 use un arreglo con dimensión (en inglés ARRAY) tal como REG().

2– Las variables de arreglo incluyen una expresión o un subíndice de una dimensión en el rango de 0 a 254, incluidos en paréntesis. Variables válidas de arreglos son:

   K(9) ARRAY(PRESION) OUT(INT(AK))**

   BASIC ampliado de FACTS Eng. no incluye arreglos de suscritos dobles tales como A(X, Y). BASIC puede representar un arreglo de dos dimensiones como un arreglo "codificado"de una dimensión. Para convertir el arreglo de dos dimensiones ARRAY(línea, columna) en un arreglo de una dimensión comience con la declaración de dimensión DIM ARRAY(línea* columna). Luego, en vez de usar la notación de índice doble B = A(I, J), use la expresión de un índice equivalente B = A(COLUMN*I+J-COLUMN). El programa STATS.ABM de BASIC (en el subdirectorio \LIBRARY) contiene un programa ejemplo de estadística que usa el concepto de un arreglo suscrito doble.

3. Las variables de string son una forma especial de variables de arreglo y son representadas por el carácter de signo del dólar y una expresión incluídos en paréntesis. La dimensión de las variables de string se extiende a partir de 0 a 254. Use la declaración STRING para asignar la memoria para las variables de string. Las variables válidas de string son:

```
1000 $(0)= "Primera variable de string"
1010 I=2
1020 $(I)= "Tercera variable de string"
1030 K24=1
1040 $(k24)=" Segunda variable de string"
```

4. Cuando es importante la velocidad de ejecución, no todas las variables son creadas iguales.
   A. El programa BASIC toma más tiempo para procesar variables con dimensión que lo que hace para procesar variables que tienen solamente caracteres. Vea el APÉNDICE D, VELOCIDAD DE EJECUCIÓN DE PROGRAMA BASIC.
   B. El programa BASIC toma más tiempo para procesar variables con muchos caracteres que lo que hace para procesar variables que tienen sólo un carácter. Vea el APÉNDICE D, VELOCIDAD DE EJECUCIÓN DE PROGRAMA BASIC.
5. Aunque no es típico de muchos programas BASIC, los nombres de variables no pueden contener cualquiera de las palabras claves que constituyen el sistema de instrucciones de

BASIC. Las variables BEND y LETOFF no podrían ser usadas puesto que estas contienen las palabras claves END y LET.

EL APÉNDICE B, PALABRAS RESERVADAS enumera todas las palabras reservadas que no se pueden usar como parte de un nombre de una variable. Como regla general, los nombres de variables sin vocales serán aceptables puesto que la mayoría de las palabras claves contienen por lo menos una vocal. Las excepciones a esto son las palabras claves CHR$, CR, DTR, RND, SGN, y SQR.

Expresiones

1. Expresiones matemáticas

Una expresión matemática es una fórmula que evalúa a un número. Una expresión puede tener operadores, números y variables. Una expresión puede simplemente ser un número o una variable o ella puede ser compleja por ejemplo **(B-SQR(B**2-4*A*C))/(2*A)**.

2– Expresiones de comparación

Una expresión de comparación es una expresión lógica que prueba la relación entre dos operandos. Las expresiones de comparación implican el uso de = (igual), < > (no igual), > (mayor que), < (menor que), > = (mayor que o igual) y < = (menor que o igual).
Una expresión de comparación puede tan simple como **IF B>20 THEN**..., o tan compleja como **COS(B**2)>SQR(SIN(C).AND.NOT(B>C)**.

3. Precedencia de operadores en expresiones

Las reglas para evaluar una expresión son simples.

Cuando una expresión se explora de izquierda a derecha, una operación no se realiza hasta que se encuentra un operador de precedencia más baja o igual. La precedencia de operadores desde más alta a más baja del BASIC es:

A. Operadores incluidos en paréntesis ()

B. exponenciación (* *)

C. Negación(-)

D. Multiplicación (*) y división (/)

E. Adición (+) y substracción (-)

F. Expresiones de comparación (=,,>,<,>=,<=)

G. AND lógico (AND.)

H. OR lógico (OR.)

I. OR exclusivo lógico (XOR.)

COMANDOS DEL SISTEMA

CAPÍTULO

3

En este capítulo...

Comandos del sistema

Todos los comandos del sistema deben ser entrados mientras el módulo ASCII/BASIC está en el modo COMMAND (COMANDO). Cualquier tentativa de incluir un comando de sistema en un programa generará un mensaje de error BAD SINTAX (sintaxis errada).

Los comandos de sistema descritos en este capítulo son:

AUTOLN
AUTOSTART
COMMAND@
CONT
DELPRM
EDIT
ERASE
LIST
NEW
PROGRAM
RENUMBER
RESET
RUN
SAVE

Se describen en las páginas a continuación cada uno de los comandos.

Los comandos son descritos en el formato siguiente:

Función	Descripción
Sintaxis	Como se escribe en el programa
Vea también	Otros comandos realcionados
Uso	Se definen los operandos y casos específicos
Exemplo	Explicados por sí mismos

AUTOLN

3

Función Entrada del número de línea de programa automática

Sintaxis AUTOLN *número de línea inicial, incremento*

Uso Use AUTOLN para entrar automáticamente números de línea cuando se hace el programa. La enumeración de líneas automática comienza con el *número de línea inicial*. Las líneas de programa sucesivas serán aumentadas por el *incremento* especificado.

El *incremento* es opcional. El valor prefijado del *incremento* es 10. Entre *CONTROL-C* para parar la enumeración automática de línea. Entre *CONTROL-D* para saltar el número de línea actualmente exhibido.

Ejemplo

```
> LIST
1000 REM Comienza el control del soldador
1010 PRINT1 $(0)
1020 IF DEBUG THEN PRINT2 $(0)
1030...

> AUTOLN 1002.2
1002 REM agregue documentación adicional
1004 REM $(0) = dirección + comando
1006 REM  ACK del soldador estará en $(1) cuando haya RETURN
1008 REM ninguna respuesta del soldador, $(1) = ""
1010 <entre Ctrl-D> para saltar este número de línea
1012 REM DEBUG=NOT(0) para supervisar la actividad del puerto 1
1014 <entre Ctrl-C> para salir de la enumeración de línea automática

READY
> LIST
1000 REM Comienza el control del soldador
1002 REM Agrega rápidamente documentación adicional
1004 REM $(0) = dirección + comando
1006 REM  ACK del soldador estará en $(1) cuando haya RETURN
1008 REM ninguna respuesta del soldador, $(1) = ""
1010 PRINT1 $(0)
1012 REM DEBUG=NOT(0) para supervisar la actividad del puerto 1
1020 IF DEBUG THEN PRINT2 $(0)
1030......
```

3

AUTOSTART

Función Selecciona el modo de funcionamiento de los módulos después de un reset

Sintaxis AUTOSTART *modo, programa, baud, MTOP*

Uso Cuando el AUTOSTART es entrado sin argumentos generará un mensaje (en inglés) que recuerda al usuario la sintaxis de AUTOSTART.

modo es un número, 0, 1, 2 o 3 que selecciona un procedimiento particular para hacer reset según lo mostrado en la tabla siguiente.

Tabla del Modo de Reset de AUTOSTART

Modo	Nombre	Descripción y procedimiento
0	EDIT (Modificar)	Pone el módulo en modo de comando después de un reset del módulo y se usa a través del desarrollo de programa. Use la velocidad de tranmisión (BAUD) almacenada y entre en el modo de COMANDO. El contenido de las tablas de variables no son colocadas en cero.
1	RUN (Coloca en cero las variables)	Hace funcionar un programa especificado después del reset del módulo. CLEAR (coloca en cero) el contenido de las tablas de variables y ejecuta el programa especificado por *program.*
2	RUN (Mantiene las variables)	Hace funcionar un programa especificado después del reset del módulo. Este modo también mantiene el contenido de las variables al fallar la energía y después de la ejecución de una declaración de GO_PROGRAM. No hace CLEAR a las tablas de variables y ejecuta el programa especificado por program.
3	EDIT (El usuario debe colocar una barra de espacio)	Después de que un reset el módulo espere que el usuario envíe un carácter de una barra de espacio al puerto de comando. El módulo configurará la velocidad de puerto a la velocidad del dispositivo que envió el carácter de barra de espacio Observe que el puerto 2 no apoya el modo 3 del AUTOSTART si el puerto 2 es el puerto del comando (será usado el modo 0 si se selecciona 3). Interpreta el primer carácter recibido como carácter de espacio para determinarse baud. Entra el modo COMMAND sin colocar a cero las variables.

El módulo se envía desde la fábrica en el modo 3 para una selección fácil de velocidad de transmisión.

baud es una expresión que especifica la tasa de transferencia de datos en la comunicación. El comando AUTOSTART no verifica que la velocidad especificada sea "válida". Las velocidades típicas son 300, 600, 1200, 2400, 4800, 9600, y 19200. La velocidad almacenada por AUTOSTART será la tasa por defecto usada para ambos puertos. La velocidad para cualquier puerto se puede cambiar en el programa usando la declaración SETPORT.

MTOP es una expresión que especifica la posición de memoria respaldada por batería que el BASIC ampliado puede usar para almacenaje de una variable. El valor prefijado para *MTOP* se define en los manuales de usuario específicos del módulo. Las direcciones de memoria de *MTOP* están disponibles para el usuario. *MTOP* es incluído para la compatibilidad hacia atrás con los programas de Intel MCS-51. Las nuevas aplicaciones normalmente no cambiarán este valor.

Ejemplo 1 **Cambiar la velocidad de transferencia de datos**

> AUTOSTART 0, 0, 9600
Mode = 0, Edit
Program = 0
Baud = 9600

Ejemplo 2 Hacer funcionar el programa 0 después de un reset sin borrar el contenido de las tablas de variables.

> AUTOSTART 2,0

Mode = 2, RUN (NO CLEAR)
Program = 0
Baud = 9600

Ejemplo 3 Hacer funcionar el programa 1 después de un reset e inicializar todas las variables a cero. Configurar la velocidad para ambos puertos inicialmente a 1200. Asignar 200 bytes de memoria para uso del usuario.

> AUTOSTART 1, 1, 1200, 65535-200

Modo = 1, RUN (CLEAR)
Program = 1
Baud = 1200

Manteniendo el valor de las variables al faltar energía

El modo 2 conserva todas las variables (incluyendo strings y variables con dimensión) durante la pérdida de energía, sin embargo, las declaraciones BASIC: CLEAR, MTOP, STRING y los comandos BASIC: RUN y NEW borrarán las tablas de variables. Por lo tanto estas declaraciones no se deben incluir en un programa al usar el modo 2.



NOTA: *:Al hacer DEBUG de un programa que use el modo 2 de AUTOSTART, use 'GOTO número de línea';"GOPRM' número del programa, número de línea';o el comando RESET para comenzar el programa.*

El arranque del módulo puede ser simplificado agregando comandos BASIC sin números de línea al principio del archivo de texto a ser transferido al módulo. Cuando es bajado el archivo el módulo ejecutará los comandos. Vea el ejemplo abajo.

Ejemplo

```
NEW:  Rem   Limpie el programa 0
DELPRM1: REM Borre el programa 1
AUTOSTART 2,1 REM Coloque el modo AUTOSTART
STRING 8001,79: REM  Asigna almacenaje de string
DIM REG(128): Rem  Dimesiona un ARRAY (arreglo)
10 REM Comienzo del programa
```

Strings Ya que el modo 2 no coloca en cero las tablas de variables, se deben asignar explícitamente strings con la declaración del modo de comando STRING .

Para asignar la memoria para las strings entre simplemente un comando de string según lo mostrado en los ejemplos de abajo o incluya el comando STRING sin un número de línea al principio del archivo de texto que contiene el programa para a ser bajado.

Si las tablas de variables son posteriormente colocadas en cero entonces se deben entrar otra vez un comando STRING.

Los contenidos de las tablas de variables serán borradas con: Ejecutando un comando RUN o NEW o las declaraciones CLEAR, MTOP o STRING.

Ejemplo

> STRING 2551, 254 asigna la memoria para 10 strings, cada uno con una longitud máxima de 254 caracteres.

> STRING 8001, 79 asigna la memoria para 100 strings, máximo de 79 caracteres cada string.

Arreglos Ya que el modo 2 no borra las tablas de variables, se debe colocar dimensión a los arreglos explícitamente con la declaración del modo de comando.

Usando una declaración DIM en un programa con el modo 2 seleccionado generará un error. Si el contenido de las tablas de variables son borradas posteriormente entonces se debe entrar otra vez un comando DIM.

Los contenidos de las tablas de variables serán borradas con: Ejecutando un comando RUN o NEW, o ejecutando las declaraciones CLEAR, MTOP, o STRING.

Ejemplo **> DIM REG(254), INP(64), OUT(32)**

COMMAND @

Función Selecciona el puerto de programación

Sintaxis COMMAND @ *puerto*

Uso el *puerto* es 1 o 2 y especifica el puerto de programación o de comando. BASIC envía todos los mensajes a y acepta solamente COMANDos del puerto especificado.

El puerto de programación o de comando por defecto es típicamente el puerto 1, vea el manual de usuario específico del módulo para verificar. AUTOSTART especifica la velocidad inicial para el puerto 1. Se puede usar SETPORT para cambiar velocidad de transacciones del puerto 1, sin embargo, después de que un reset se usa el valor de AUTOSTART.

La velocidad por defecto para el puerto 2 es 9600. Después de un reset, será usada la velocidad especificada por la última declaración de SETPORT. El puerto 2 no apoya el modo 3 de AUTOSTART (será usado el modo 0 si se selecciona 3).

Use el COMANDO @ para DEBUG de comunicaciones con un dispositivo externo conectado con el puerto opuesto. Se puede usar el COMMAND @ para conseguir la utilización completa de ambos puertos mientras que reduce al mínimo la necesidad de intercambio de cable o uso de cajas de conmutación.

Ejemplo 1 Asuma que se ha completado un programa para una impresora conectada con el puerto 2 para reportaje de diagnósticos de un turno. Ahora se desea hacer funcionar un regulador de un motor paso a paso usando el puerto 1. Para comenzar a programar el control del motor paso a paso:

> SETPORT 2, 9600 configura la velocidad para el puerto 2
> COMMAND@2 el puerto de programación ahora es el puerto 2

Mueva el cable de programación del dispositivo desde el puerto 1 al puerto 2. Para ir de nuevo a la programación en el puerto 1, entre COMMAND@1

Ejemplo 2 El puerto de programación se puede seleccionar en el programa para realizar cambios de programación remotos ***con el módem opcional construido en el teléfono del puerto 2. (Check with MAC)***

SYSTEM(7)=0: REM Selecciona Puerto 1 para programar
SYSTEM(7)=NOT(0): REMSelecciona Puerto 2 para programar

Vea el ejemplo de aplicación TELESERV.ABM en el directorio *ABMCOMMANDER PLUS ABM\ABM-TM*

Este comando es algo diferente en el módulo F0-CP128

3

CONT

Función Reanuda la ejecución de programa.

Sintaxis CONT

Vea también TRACE,STOP

Uso CONT se utiliza típicamente durante DEBUG (eliminación de errores) del programa. Si la ejecución de programa es parada tecleando un < *CTRL-C* > o por la declaración STOP, CONT entonces hace que se reanude la ejecución del programa donde fue parado. Antes de reanudar la ejecución del programa, los valores de variables pueden ser exhibidos o pueden ser cambiados. Si se cambia el programa, éste no puede ser continuado.

Ejemplo

```
> 10  DO
> 20 I=I+1: PRINT1 I
> 30 WHILE I>0
> RUN
1
2
3
4
5
6 (Apriete las teclas CTRL-C)

STOP - EN LA LÍNEA 30
READY
> I=-1

> 0 CONT
0
READY
 >
```

3

DELPRM

Función Borra un programa almacenado

Sintaxis DELPRM *número de programa*

Uso el *número de programa* es el número del programa almacenado que se suprimirá. Los números de programa son asignados por el comando SAVE.

Los números corresponden a la orden en la cual los programas fueron archivados. Después de que se haya hecho con éxito la operación de borrar el programa, se muestra el número de programas almacenados y la cantidad de bytes de almacenaje de programas restantes.

El número de programas almacenados no incluye el programa 0.

DELPRM se puede utilizar para borrar cualquiera de los programas almacenados.

Tecleando DELPRM 0 tiene el mismo efecto que teclear NEW.

Ejemplo

```
READY
 > DELPRM 4
7 stored programs, 28381 bytes free
>
```

EDIT

Función Lleva un programa archivado al PROGRAMA 0 para modificar (o EDIT).

Sintaxis EDIT

Uso El comando EDIT se usa para copiar el programa corrientemente seleccionado en el archivo del programa al PROGRAMA 0 para modificar.

EDIT ejecuta un comando NEW antes de copiar el programa.

Seleccione un programa del archivo del programa con el comando PROGRAM.

El programa original estará aún almacenado en el archivo de programa.

Para borrar el programa original, use DELPRM.

Ejemplo

```
> PRM1

PRM1
READY
 > LIST
10 REM  Ejemplo de comando EDIT
20  FOR X=1 TO 5
30 PRINT1 "HOLA"
40 NEXT X
PRM1
READY
> EDIT

PRM0
READY
> LIST
10 REM Ejemplo de comando EDIT
20  FOR X=1 TO 5
30 PRINT1 "HOLA"
40 NEXT X

PRM0
READY
> 20 FOR X=1 TO 10: REM Aquí se cambia la línea 20
> LIST
10 REM  Ejemplo de comando EDIT
20 FOR X=1 TO 10
30 PRINT1 "HOLA"
40 NEXT X
```

ERASE

Función Suprimir o borrar un rango de números de línea en el programa

Sintaxis ERASE *número de línea inicial, número de línea final*

Uso Se usa el comando ERASE de remover líneas de programa desde el *número de línea inicial* al *número de línea final*.



Nota: *Recomendamos guardar el programa al disco o archivar una copia en el archivo de programa antes de hacer un cambio de programa importante con ERASE puesto que no hay comando de "UNERASE "(para deshacer lo que ha sido hecho).*

Ejemplo

```
> LIST
10 REM ejemplo de BORRAR o ERASE
20 SI DEBUG THEN GOSUB 200
30 INLEN2=0
40 INLEN1=0
50 REM comienzo del programa principal

> ERASE 10, 40
> LIST
50 REM Comienzo del programa principal
```

LIST

Función Lista o muestra el programa seleccionado.

Sintaxis LIST *línea de comienzo, línea final*

Uso El comando LIST se usa para mostrar el programa actualmente seleccionado. LIST inserta espacios en el programa después de los números de línea y antes y después de instrucciones para mejorar el aspecto y la legibilidad del programa. LIST se puede usar de cuatro maneras:

- LIST exhibe el programa entero.
- LIST *línea de comienzo* muestra el programa desde la *línea de comienzo* hasta el final del programa -
- LIST *línea de comienzo, línea de comienzo* muestra solamente una sola línea de programa.
- LIST *línea de comienzo, línea final* muestra las líneas desde la *línea de comienzo* y termina con la *línea final*.

Ejemplo

```
> LIST
10 REM EJEMPLO LIST
20 A=B : C=2
30 END

READY
> LIST 20
20 A=B: C=2
30 END

READY
> LIST 10,20
10 REM: EJEMPLO LIST
20 A=B : C=2

READY
>LIST 20,
20 A=B: C=2
```

Un listado causado por LIST puede ser terminado tecleando < *CTRL-C* >. LIST puede también ser parado apretando XOFF < *CTRL-S* > y después ser recomenzado incorporando XON < *CTRL-Q* >.

Observe que la única entrada por el puerto serial que aceptará BASIC después de XOFF es XON o < *CTRL-C* >.

Vea SETPORT para más información sobre el control de flujo XON y XOFF .

NEW

Función Borra el PROGRAMA 0 y las variables.

Sintaxis NEW

Uso El comando NEW se usa para borrar el PROGRAMA 0 en la memoria de datos. NEW también borra todas las variables. No hay ningún comando"UNDO NEW" de modo que use NEW con precaución.

Ejecutar NEW tiene el mismo efecto que DELPRM 0.

PROGRAM o PRM

Función Selecciona un programa archivado

Sintaxis PROGRAM *número*

Uso El comando PROGRAMA se usa para seleccionar un programa para el LISTado o para hacerlo funcionar. El *número* especifica qué programa desea tener acceso el usuario (vea SAVE).

Si se hace una tentativa de seleccionar un número de programa que sea mayor que el número de programas almacenados en el archivo del programa o menos que 0 entonces será generado el mensaje de ERROR: PROGRAM ACCESS. Este mensaje de error también será generado si se hace una tentativa de realizar cambios a un programa en la memoria de programa.

3

RENUMBER

Función RENUMBER un rango de líneas de programa

Sintaxis RENUMBER *comenzar, terminar, incremento, comenzar de nuevo*

Uso RENUMBER le permitirá agregar líneas de programa a una sección del programa donde previamente no había lugar para colocar mas lineas.

Todas las líneas de programa desde la línea de programa *comenzar* hasta la línea de programa *terminar*, y todas las referencias a estas líneas dondequiera en el programa se le asignará un nuevo número de línea.

incremento es opcional y especifica la diferencia entre números de línea consecutivos. *incremento* por defecto tiene un valor de 10.

Si un número de línea *comenzar de nuevo* es definido entonces se especifica que el programa entero es vuelto a numerar por la cantidad del incremento. La línea *comenzar de nuevo* es el primer número de línea del programa con un nuevonúmero. En este caso, los parámetros *comenzar* y *terminar,* se ignoran pero se deben incluir en el comando RENUMBER.

Ejemplo

```
> LIST
10 STRING 8001, 79: REM STRING hace un CLEAR
20 DEBUG = 0
30 LOCKOUT = 0
40 BREAK= 1
50 IF NOT DEBUG THEN GOTO 101
100 GOTO INLEN1 = 0
101 DIM REG(20)
102 FOR I = 1 TO 20
103 REG(I) = 1
104 I NEXT
200. . .

> RENUMBER 100, 104, 5
> LIST
10 STRING 8001, 79: REM STRING hace CLEAR
20 DEBUG = 0
30 LOCKOUT= 0
40 BREAK= 1
50 IF NOT DEBUG THEN GOTO 105
100 GOTO INLEN1 = 0
105 DIM REG(20)
110 FOR I = 1 A 20
115 REG(I) = 1
120 NEXT I
200 . . . . .
```

RESET

Función Ejecutar un reset del software

Sintaxis RESET

Uso El RESET ejecuta las mismas rutinas de inicialización del software ejecutadas después de que haya ocurrido un reset del hardware. El RESET se puede ejecutar en el modo COMMAND para verificar la operación de AUTOSTART.

RUN

Función Coloca el contenido de las tablas de variables en cero y ejecuta el programa seleccionado

Sintaxis RUN

Uso Entrando RUN hace que BASIC coloque todas las variables en cero, interrupciones en cero, reset stacks y comienza la ejecución del programa corrientemente seleccionado desde el primer número de línea.

La ejecución de programa puede ser parada con STOP enviando al módulo un carácter < *CTRL-C* >. Para deshabilitar la función < *CTRL-C* > vea la declaración BREAK. RUN comienza siempre la ejecución con el número de línea de programa más bajo.



NOTA: Si usted está usando el modo 2 del AUTOSTART, entonces no use el comando RUN para comenzar el programa. Para comenzar la ejecución sin borrar las variables o en un cierto punto en el programa (con excepción del principio), use GOTO.

Ejemplo

```
> 10 FOR J=1 TO 3:PRINT1 J:NEXT J
> 20 PRINT1 "COMIENZO"
> RUN
1
2
3
COMIENZO

READY
>
```

SAVE

3

Función Almacena (o archiva) el programa seleccionado en el archivo de programa

Sintaxis **SAVE**

Uso El comando SAVE se usa para almacenar el programa actualmente seleccionado, en la memoria de datos (PROGRAMA 0) o en el archivo del programa, en el espacio libre siguiente en el archivo del programa. Los programas se almacenan secuencialmente en el archivo de programa. Cada vez que se ejecuta el comando SAVE, la cantidad y (el número) de programas archivados será aumentado en uno. La cantidad de programas almacenados es limitada solamente por el tamaño de los programas y de la cantidad de memoria de almacenamiento de programa disponible. Cuando se entra el comando SAVE, BASIC mostrará el número de archivo del programa en la pantalla. Este número es usado por PROGRAM y GO_PROGRAM para recuperar o para ejecutar un programa en el archivo.

Ejemplo

```
> PRM 0
> LIST
10 PRINT1 "PROGRAMA DE PRUEBA"
> SAVE
Saving Program 7
7 Stored programs, 51154 bytes program storage bytes free

> GOPRM 7
PROGRAMA DE PRUEBA
> NEW borra el programa 0
> PRM 0
> LIST   Si, se borró....
> PRM 7 selecciona el PROGRAMA  7 otra vez
> EDIT lo mueve al PROGRAMA 0

> PRM 0
READY

> LIST
10 PRINT1 "PROGRAMA DE PRUEBA"
```

El PROGRAMA 0 fue almacenado en el archivo de memoria de programa como PROGRAMA 7. Luego el programa almacenado fue recuperado para modificar o EDIT.

Vea DELPRM para borrar un programa almacenado en el archivo de programa.

DECLARACIONES

CAPÍTULO

4

En este capítulo...

4

Capítulo 4: Declaraciones

Declaraciones

Propósito de este capítulo

Este capítulo contiene un listado alfabético de las declaraciones y de los operadores de BASIC ampliado de FACTS Eng. que se ofrecen en todos los tipos de módulos BASIC. El manual de usuario específico del módulo describe cualquier diferencia de definición de estas declaraciones así como cualquier característica específica a un módulo particular.

4

Las declaraciones son descritas en el formato siguiente:

Declaración   Tipo de declaración

Los comandos son descritos en el formato siguiente:

Función	Descripción
Sintaxis	Como se escribe en el programa
Vea también	Otros comandos realcionados
Uso	Se definen los operandos y casos específicos
Exemplo	Explicados por sí mismos

Manual de referencia de BASIC de FACTS Eng. 4–5

4

ABS - Operador matemático

Función Calcula el valor absoluto de la expresión.

Sintaxis ABS(*expresión*)

Uso Devuelve el valor absoluto de la *expresión*.

Ejemplo
```
PRINT ABS(5)
5
PRINT ABS(-5)
5
```

ASC - Operador de string

Función Cambia o devuelve el código ASCII de un carácter en un string

Sintaxis código = ASC (*variable de string , posición*)
ASC(*variable de string , posición*) = *código*

Vea también CHR$

Uso El operador ASC devuelve el *código* ASCII (0-255) para el carácter en cualquier posición específica en *variable de string*.

La declaración ASC asigna el código ASCII (0-255) al carácter en cualquier posición específica en *variable de string*.
El rango válido para la *posición* es 1 a 255.

Normalmente las funciones de administración de strings MID$, INSTR, LEFT$ y RIGHT$ serían usadas para manipular strings. Un caso especial existe cuando se deben manipular dentro de strings los caracteres de control ASCII **Null** (< Ctrl-@ >, ASCII = 0) y CR, **retorno de carro**, (< Ctrl-M >, ASCII = 13). Estos caracteres son usados como delimitadores por toda declaración que manaja strings excepto ASC. Por lo tanto, solamente puede ser usada la función ASC para procesar strings con los caracteres NULL y **retorno de carro** dentro del string.

BASIC agrega un carácter de retorno de carro (ASCII, 13) para identificar el final o terminación de un string.

Ejemplo
```
 $(0)="123"

> ASC($(0), 2)=65:  REM Código ASCII para "A"
> P. $(0) 1 TO 3

> P. $(0)
1A3

> P. ASC($(0), 3), SPC(2), CHR$(ASC($(0), 3))
51 3
```

ATN - Operador matemático

Función Calcula el arcotangente de la expresión

Sintaxis ATN (e*xpresión*)

Uso Calcula el arcotangente de la *expresión*. El resultado está en radianes. Los cálculos se realizan con 7 dígitos significativos. El operador ATN devuelve un resultado en el rango - PI/2 (-3,1415926/2) y PI/2.

Ejemplo

```
PRINT  ATN(3.1415926)
1.2626272
```

```
PRINT ATN(1)
.78539804
```

BIT y BITS - Entradas - salidas

Función Descifra y codifica una variable de 16 bits

Sintaxis *var* = BIT(*subíndice*)

BIT(*subíndice*) = *expresión*

var = BITS

BITS = *expresión*

4

Vea también PICK

Uso El BIT será usado normalmente para descifrar puntos del estado de entradas del PLC o para codificar salidas del PLC. El *subíndice* se refiere a una posición de bit particular (0) a (15) o S para los 16 bits.

Ejemplo

```
PRM 0
READY
> LIST
10 REM
20 WRD=65534: REM WRD es una variable que contiene un valor
30 BITS=WRD: REM de 16 bits  de un registro de entradas y salidas
30 FOR BT=0 TO 15
40 PRINT1 "BIT(", BT,") - >",
50 IF BIT(BT) THEN PRINT1 "ON" ELSE PRINT1 “OFF”
60 NEXT BT
PRM 0
READY
> RUN
BIT(0) - > OFF
BIT(1) - > ON
BIT(2) - > ON
BIT(3) - > ON
BIT(4) - > ON
BIT(5) - > ON
BIT(6) - > ON
BIT(7) - > ON
BIT(8) - > ON
BIT(9) - > ON
BIT(10) - >ON
BIT(11) - >ON
BIT(12) - >ON
BIT(13) - >ON
BIT(14) - >ON
BIT(15) - >ON
```

BREAK - Control de flujo

Función Permitir e deshabilitar < *CTRL-C* > para parar la ejecución del programa.

Sintaxis BREAK= *Verdadero o falso*

Vea también LOOKOUT, CONT

Uso *Verdadero o falso* es una expresión que cuando es igual a 0 deshabilita la terminación del programa al apretar las teclas< *CTRL-C* >. Si *verdadero o falso* no es 0 entonces BASIC verifica durante INPUT y después de ejecutar cada línea de programa para considerar si se ha entrado un <*CTRL-C*>. Un programa interrumpido con < *CTRL-C* > se puede reanudar en el punto de interrupción con CONT.



NOTA: verdad = cero, falso = diferente a cero

Normalmente, se permite usar BREAK durante el desarrollo de un programa solamente. BREAK debe ser deshabilitada para la mayoría de aplicacioness industriales.

Si BREAK es deshabilitada por BASIC entonces la ejecución del programa continuará hasta llegar a una declaración END o STOP o hasta que se genera un error.

El modo 1 o 2 del AUTOSTART ejecuta un programa después de un reset. Si el programa hace que BREAK deshabilite y entonces nunca ejecute una declaración END, entonces el programa no tendrá más acceso normalmente. Para tener acceso al programa, usted debe cambiar el modo de AUTOSTART. Vea el manual de usuario específico del módulo para la localización del puente CLEAR ALL que puentea los parámetros almacenados de AUTOSTART.

Ejemplo

```
> REM permite la interrupción del programa
> 10 BREAK = 1

>  REM deshabilita la interrupción del programa
> 10 BREAK = 0

> REM si< CTRL-C > está activado entonces lo deshabilita
> 10 IF BREAK THEN BREAK = 0

> REM si< CTRL-C > no está activado entonces lo habilita
> 10 IF NOT(BREAK) THEN BREAK = 1
```

4

BYTE - Operador avanzado

Función Leer o escribir un valor de un byte en memoria variable de almacenaje

Sintaxis *variable* = BYTE (*dirección*)

BYTE (*dirección*) = *Byte de datos*

Vea también LOAD@, STORE@, WORD

Uso La declaración BYTE se usa para recuperar o para asignar un *byte de datos*, en el rango de 0 a 255, a una *dirección* en la memoria variable de almacenaje, respaldada por batería. El rango de *direcciones* válidas es específico al módulo. El rango de memoria reservado para uso por el intérprete de BASIC no debe ser escrito por el usuario. Un alternativa a usar el modo 2 del AUTOSTART para conservar datos con falta de energía es almacenar un valor en memoria de datos. Para hacer esto, el usuario debe asignar una cierta memoria libre (memoria no usada por BASIC) para que se almacenen los datos. Use el operador MTOP, descrito más adelante en este capítulo, para suministrar al usuario memoria libre para el almacenaje de datos.

Ejemplo Almacenar una string en MTOP

```
> MTOP=16383 asigna 16K de memoria de datos. Esto es equivalente
  a 16384 registros de datos retentivos de 8 bits del PLC .
> AUTOSTART modo, programa, baud,  16383
        almacena el nuevo valor para MTOP
> 30 STRING 100,10:  REM asigna la memoria para los strings
> 40 INPUT "CUÁL ES SU NOMBRE? ", $(1): REM String de entrada
> 50 I=D0: I=I+1
> 60 BYTE(16385+I)=ASC($(1), I): REM almacena códigos ASCII
> 70 UNTIL ASC($(1), I)=13: END: REM Fin de entrada
> 80 BYTE(16384)=I-1
> 90 REM almacena la longitud del string para señalar el proximo
            libre
> 91 REM localización de memoria
```

CALL - Operador avanzado

Función CALL invoca un subrutina assembler o lenguaje de máquina

Sintaxis CALL *dirección*

Uso La declaración de CALL se usapara llamar un programa del lenguaje assembler. la *dirección* es la posición de memoria inicial de la rutina del lenguaje assembler. Las rutinas del lenguaje assembler se deben situar en memoria de programa.

Los usuarios que entienden la arquitectura y el lenguaje assembler de la familia y el microcontrolador de Intel MCS-51 para agregar una función creada especialmente deben consultar la fábrica para ayuda adicional de información y de la aplicación.

Generalmente, todos los recursos de sistema e interrupciones son usados por BASIC. Las rutinas llamadas deben hacer PUSH/POP todas las direcciones de memoria internas usadas. La mayoría de los productos permiten bajar programas de assembler o de la lenguaje 'C' a la memoria RAM repaldada por batería. Esto permite la adición de funciones creadas por el usuario sin “quemar” o colocar datos en un EPROM.

CBY - Operador avanzado

Función Leer el contenido de la dirección de memoria en la memoria de almacenaje de programa

Sintaxis variable = CBY (*dirección*)

Uso El operador CBY se puede usar para asignar el contenido de las posiciones de memoria individuales en memoria de programa a una variable. Ya que la memoria de programa no se puede escribir directamente, no se le puede asignar un valor al operador de CBY.

CHR$ - Operador de string

Función Convierte un código ASCII en una sola cadena de caracteres o un "string"

Sintaxis *variable de string* = CHR$(*código*)

Vea también ASC

Uso CHR$ devuelve una cadena de caracteres única que corresponde a un código ASCII. CHR$ es útil para crear strings que contienen los caracteres ACII que no se imprimen y los caracteres que no se pueden entrar desde el teclado.

código debe estar en el rango 0 a 255.

Ejemplo Secuencia de escape ANSI usando CHR$

```
> $(0)=CHR$(27)+"[2J" limpia la pantalla
```

Especial El carácter delimitador de string **retorno de carro** (código ASCII = 13) y **null** (código ASCII = 0) se pueden imprimir (PRINT) con CHR$. CHR$ muestra una longitud de string cero para estos códigos.

Use el operador ASC para entrar caracteres **null** y **retorno de carro** dentro de una cadena de texto. Use PRINT USING (\count \) para imprimir cadenas de texto con caracteres **retorno de carro** y **null** embutidos.

Ejemplo

```
> PRINT "muestre una línea repetidamente", CHR$(13), "
      PRINT" una línea repetidamente

> $(0)=CHR$(13)
> PRINT LEN($(0))
0
```

CLEAR - Control de flujo

Función CLEAR borra el contenido de las tablas de variables

Sintaxis CLEAR

Vea también CLEAR I, S CLEAR

Uso La declaración CLEAR se usa para configurar todas las variables incluyendo strings y arreglos (arrays) con dimensión a cero, para deshabilitar interrupciones, para resetear STACKS de control y para cancelar la declaración ONERR.

La declaración CLEAR no afecta el temporizador de software, el reloj de calendario respaldado por batería o resetea la memoria asignada por la declaración STRING.

CLEAR I - Interrupción

Función Deshabilita interrupciones de programa

Sintaxis CLEAR I

Vea también S CLEAR, CLEAR S

Uso La declaración CLEAR I resetea el bit de interrupción BASIC e deshabilita las interrupciones de programa permitidas por las declaraciones ONTIME y ONPORT.

Esta declaración se puede usar para prevenir interrupciones durante ciertas secciones de un programa BASIC.
ONTIME y ONPORT deben ser ejecutados otra vez antes de que sean activadas las interrupciones respectivas.

CLEAR S- Control de flujo

Función Resetea los stacks de control y de argumentos

Sintaxis CLEAR S

Vea también CLEAR, CLEAR I

Uso La declaración CLEAR S se usa para resetear stacks de control y de argumentos BASIC. Esta declaración se puede usar para salir prematuramente de un subrutina (GOSUB-RETURN) o de un lazo de control FOR-NEXT,-DO-UNTIL o DO-WHILE. Después de ejecutar una declaración CLEAR S el usuario usará normalmente una declaración GOTO para saltar de nuevo al cuerpo principal del programa.

CLEAR S cancela todas las rutinas de GOSUB y lazos de control.

Ejemplo

```
> 10 PRINT1 "Prueba de multiplicación"
> 30 INPUT "¿Cuántos segundos desea Ud.? ",S
> 40 ONTIME S, 200
> 50 FOR I=2 TO 9
> 60 N=INT((RND)*10)
> 70 PRINT1 N, "*",i, "=? ",
> 80 TIME=0
> 90 INPUT, $(0)
> 95 R = VAL($(0))
> 100 IF R<>N*I THEN PRINT1 "INCORRECTO": GOTO 60
> 110 PRINT1  "ESO ES CORRECTO!"
> 130 NEXT  I
> 140 PRINT1: PRINT1 "ESO ES TODO"
> 150 END
> 200 CLEAR S
> 210 PRINT1 "USTED TOMÓ MUCHO TIEMPO"
> 220 GOTO  20
```

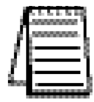
COMERR - Operador avanzado

Función Bit de error de la función CRC-16 (Verificación de error)

Sintaxis *verdad o falso* = COMERR *puerto de acceso*

Uso Si está activada la verificación de error CRC-16, COMERR será *verdadero* cuando se ha detectado un error CRC-16 después de una declaración INPUT1, INPUT2, o INPUT3. COMERR es *falso* cuando el CRC-16 calculado es igual a los dos caracteres CRC-16 recibidos.

Si se encuentra un error de paridad, el carácter es ignorado (causando un error CRC-16). COMERR se hace OFF (*falso*) en el comienzo de una declaración INPUT.



NOTA: verdadero = cero , falso = diferente a cero

Ejemplo Vea el CAPÍTULO 8, AVANZADO

4

COPY - Gerencia de memoria

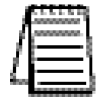
Función Copia un bloque de memoria del módulo BASIC

Sintaxis COPY *dirección inicial, dirección final, dirección de destino*
COPY *dirección inicial, K(número de bytes), dirección de destino*

Uso Copia un bloque de memoria del módulo BASIC comenzando en la *dirección inicial* de la fuente hasta la *dirección final* a la memoria del módulo BASIC comenzando en la *dirección de destino*.

Opcionalmente, se puede especificar el *número de bytes* de memoria para copiar como una expresión en paréntesis, a continuación de "K".

El tamaño de bloque máximo que puede ser copiado es 65535 bytes.



NOTA*: Vea el manual de usuario específico del módulo para el mapa de memoria del módulo que usted está usando.*

Ejemplo Este ejemplo crea tablas de variables de string que se almacenan en la parte superior del banco 1 del almacenaje de datos. COPY se usa para almacenar y para recuperar las tablas de variable de string. Esto es útil cuando se necesitan más de 254 variables de string o cuando necesita ser reducida la cantidad de memoria asignada para los strings.

```
1000 REM
1010 REM INTERCAMBIO DE VARIABLES DE STRING
1020 REM AMPLIA EL NÚMERO DE LAS VARIABLES de STRING
1030 REM PARA AHORRAR MEMORIA, LOS STRINGS SE PUEDEN
1035 REM DEFINIR  LITERALMENTE DURANTE EL PROCESO DE
1040 REM TRANSFERENCIA DEL PROGRAMA Y GUARDADO
1045 REM  EN MEMORIA  USANDO COPY.
1050 REM ASIGNA LA MEMORIA
1060 REM  PARA 10, 254 STRINGS
1070 STRING 2551,254
1080 REM Cantidad de bytes en un banco de memoria
1090 BANK=65535
 1100 REM Cantidad total de bytes de espacio de almacenaje de string
1110 SIZE=MTOP-WORD(104H)
1120 REM Comienzo del espacio de almacenaje de string
1130 TBL(0)=WORD(104H)
1140 REM Dirección inicial de la tabla de string 1,al final del banco 1
1150 TBL(1)=BANK*2+1-SIZE
1160 REM Dirección inicial de la tabla de string 2, al final del banco 1
1170 TBL(2)=BANK*2+1-SIZE*2
1180 REM HACE ARREGLO DE STRING DE PRUEBA, TABLA 1
```

```
1190 FOR I=0 TO 9
1200 FOR J=1 TO 254
1210 ASC($(I), J)=I+48
1220 NEXT J
1230 PRINT2 $(I)
1240 NEXT I
1250 REM ARCHIVA LA TABLA DE STRING 1
1260 COPY TBL(0), K(SIZE), TBL(1)
1265 REM HACE ARREGLO DE STRING DE PRUEBA, TABLA 2
1270 FOR I=0 TO 9
1280 FOR J=1 TO 254
1290 ASC($(I), J)=I+65
1300 NEXT J
1310 PRINT2 $(I)
1320 NEXT I
1330 REM ARCHIVA LA TABLA DE STRING 2
1340 COPY TBL(0), K(SIZE), TBL(2)
1350 REM OBTIENE LA TABLA DE STRING 1
1360 COPY TBL(1), K(SIZE), TBL(0)
1365 REM IMPRIMA TABLA DE STRING 1
1370 FOR I=0 TO 9
1380 PRINT2 $(I)
1390 NEXT I
1400 REM OBTIENE LA TABLA DE STRING 2
1410 COPY TBL(2), K(SIZE), TBL(0)
1415 REM  IMPRIMA TABLA DE STRING  2
1420 FOR I=0 TO 9
1430 PRINT2 $(I)
1440 NEXT I
```

COS - Operador matemático

Función Calcula el coseno de la expresión

Sintaxis COS(*Expresión*)

Uso Calcula el coseno de la *expresión*; la *expresión* está en radianes. Los cálculos se realizan con 7 dígitos significativos. La expresión debe estar entre + 0 -200000.

Ejemplo

```
PRINT COS(3.14/4)
.7071067

PRINT COS (0)
1
```

CR - Entradas - salidas

Función Usado en la declaración PRINT para generar un **retorno de carro**

Sintaxis PRINT CR,

Vea también PRINT, SPC, TAB, USING @

Uso La función CR es una abreviatura para CHR$(13). El CR causará que sea enviado un carácter de **retorno de carro** (ASCII=13) al puerto serial cuando se encuentra en la declaración PRINT. No se enviará ningún carácter de **avance de línea**.

La función CR parecerá que no trabaja correctamente con las impresoras y los terminales que agregan automáticamente un carácter **avance de línea** cuando se recibe el carácter de **retorno de carro**. La función CR se puede usar para actualizar en varias ocasiones la misma línea en un CRT.

Ejemplo

```
> 10 FOR J=1 TO 100
> 20 PRINT1 USING(###), J, CR,
> 30 NEXT J
```

Programa equivalente usando CHR$

```
> 10 FOR J=1 TO 100
> 20 PRINT1 USING(###), J, CHR$(13),
> 30 NEXT J
```

DATA - Entradas - salidas

Función DATA especifica expresiones para las declaraciones READ

Sintaxis DATA, *expr, expr,*....

Vea también READ, RESTORE

Uso *expr* es una expresión o una constante numérica. DATA declara las expresiones que se pueden asignar a las variables que siguen la declaración READ.

Las múltiples expresiones especificadas en una declaración DATA son separadas por comas. Las declaraciones DATA se pueden poner en cualquier lugar en el programa puesto que no se ejecutan. Las declaraciones DATA se unen en la orden que aparecen en el programa.Es decir, se puede colocar mas de una declaración DATA y los datos siguen en secuencia.

Ejemplo

```
10 REM Cargue el arreglo (ARRAY)  ARREG con constantes
20 FOR D=1 TO 5
30 READ ARREG(D)
40 NEXT D
50 DATA 35, -1, 10E3, 0FFEH, ARREG(2)*2+D
```

Por lo tanto, el arreglo tendrá los siguientes valores después de ejecutar este programa:

ARREG(1) = [35]

ARREG(2) = [-1]

ARREG(3) = [10E3]

ARREG(4) = [0FFEH]

ARREG(5) = [3]

DATE$ - Operador de string

Función DATE$ configura y recupera la fecha del calendario respaldado por batería,

Sintaxis DATE$ = *expresión de string*
variable de string = DATE$

Vea también TIME$

Uso Cuando está ajustada a formato correctamente, la *expresión de string* configura el año, el mes, el día del mes y el día de la semana del calendario respaldado por batería. La *expresión de string* debe estar en una de las formas siguientes:

mm-dd-aa-s (por ejemplo. DATE$ = "9-25-88-7")
mm/dd/yy/w

variable de string es una variable de string que contiene la fecha del calendario respaldado por batería devolvida por DATE$. DATE$ devuelve una string de longitud variable en la forma mm-dd-aa.

El reloj del calendario respaldado por batería tiene una precisión de +/- 1 minuto por mes con el módulo a 24 grados C.

Ejemplo

```
> DATE$ = "10/10/88/1"
> PRINT1 DATE$
Monday 10-10-88

> DATE$ = "2-29"
> PRINT1 DATE$
Monday 02-29-88
> DATE$ = "1-1-90/4"
> P. DATE$
Thursday 01-01-90

> PRINT1 "El año es 19"+MID$(DATE$, LEN(DATE$)-1)
El año es 1990

> MON_POS = INSTR(DATE$, "")+1
> DAY_POS = INSTR(DATE$, "-")+1
> P. "Estamos en el mes";VAL(MID$(DATE$, MON_POS, 2))
Estamos en el mes 1
```

DBY - Operador avanzado

Función Escribe a las direcciones de memoria especiales (memoria interna de la CPU)

Sintaxis *variable* = DBY (*dirección*)
DBY (*dirección*) = *byte de datos*

Vea también SYSTEM

Uso DBY se usa para recuperar o asignar un *byte de datos* a uno de los 256 bytes de memoria especial dentro del módulo BASIC. La *dirección* debe estar entre 0 y 255 (0FFH) incluyendo estos números.



***NOTA**: Las posiciones de memoria direccionadas por el operador DBY son reservadas para uso del intérprete BASIC y pueden haber cambiado con nuevas revisiones de firmware. En lo posible, se debe usar la declaración SYSTEM equivalente.*

Resumen del uso de DBY

MEMORIA LOCALIZACIÓN	USO de BASIC
DBY(21)	La cuenta de carácter **null** configurada por el comando NULL
DBY(23)	El formato de la declaración PRINT configurado por USING
DBY(24)	Dirección de memoria del programa de destino menos un para la declaración PGM - byte menos significativo. También la base de tiempo DELAY.
DBY(25)	Dirección de información fuente para la declaración PGM - byte menos significativo. También byte menos significativo del CRC-16
DBY(26)	Dirección de memoria del programa de destino menos un para la declaración de PGM - byte más significativo . También byte más significativo Crc-16
DBY(27)	Dirección de la información de la fuente para la declaración de PGM - byte más significativo . También conteo del carácter RTS OFF-delay para el "handshake" de hardware.
DBY(30)	Número de bytes a escribir en la declaración PGM - byte menos significativo
DBY(31)	Número de bytes a escribir en la declaración PGM - byte más significativo
DBY(35)	Apaga o enciende la verificación de error CRC-16
DBY(41)	DBY(41)=NOT(0) = COMMAND@2 DBY(41)=0 = COMMAND@1
DBY(71)	Porción fraccionaria de TIME

DELAY - Misceláneo

Función Inserir una pausa

Sintaxis DELAY *milisegundos*

Vea también ONTIME, TIME

Uso *milisegundo* es una expresión que indica el número de milisegundos que BASIC ampliado debería detenerse brevemente antes de ejecutar la próxima declaración.

Use DELAY en vez de lazos de software en los programas que funcionan apropiadamente en el hardware futuro que tendrá velocidades de clock mayores que lo que era posible.

Ejemplo

```
10 PRINT2 "Está usted despierto?"
20 DELAY 1000: REM se detiene por 1 [s] para una respuesta
30 IF INLEN2=0 THEN GOTO 60
40 PRINT1 "El dispositivo en el puerto 2 está vivo"
50 END
60 PRINT1 "El dispositivo en el puerto 2 está durmiendo"
70 END
```

4

DIM - Gerencia de memoria

Función Asigna la memoria para arreglos (arrays) numéricos

Sintaxis DIM *Var(expr)* , *var(expr)...*

Uso DIM declara variables de arreglo que no sean strings y asigna espacio en la memoria de datos para su almacenaje.

expr especifica el número de elementos o de subíndices en el arreglo y debe ser menor de 255. Los arreglos BASIC deben ser de una dimensión.

Para implementar un arreglo de dos dimensiones Vea ARREGLOS DE ÍNDICES DOBLES en el capítulo COMO COMENZAR.

La declaración DIM puede aparecer en cualquier lugar en el programa excepto antes de una declaración STRING.

Al tratar de redimensionar un arreglo o tener acceso a un elemento de arreglo que esté fuera del alcance de la variable con dimensión generará un mensaje, ERROR: ARRAY SIZE -SUSCRIPT OUT OF RANGE - IN LINE XX (TAMAÑO DEL ARREGLO - SUBÍNDICE FUERA de RANGO - EN LA LÍNEA XX).

Puesto que el modo 2 del AUTOSTART conserva datos en arreglos con dimensión, se debe ejecutar la declaración DIM en modo COMMAND. No re-dimensione arreglos en el programa al usar el modo 2 de AUTOSTART.

Si se usa una variable puesta en un arreglo que no ha sido dimensionada por la declaración DIM entonces BASIC dimensiona automáticamente el arreglo con 10 elementos.

Es una buena práctica dimensionar explícitamente todos los arreglos.

Ejemplo

```
10 DIM A(20)  B(20)
20 C(2)=9: REM ARREGLO DE TAMAÑO ASIGNADO  DE 10
30 DIM D(C(2)*2)
40 REM LA EXPRESIÓN C(2)*2 DEBE SIEMPRE SER < = 254
```

DO-UNTIL - Control de flujo

Función Ejecuta un lazo hasta que una prueba en la parte inferior del lazo sea TRUE

Sintaxis DO-UNTIL *expresión de comparación*

Vea también DO WHILE

Uso Las declaraciones DO-UNTIL permiten que el usuario ejecute en varias ocasiones las líneas de programa entre declaraciones DO hasta UNTIL.

La *expresión de comparación,* cuando es igual a cero, representa FALSO y continúa la ejecución de instrucciones en el lazo; si no es así, termina la ejecución de instrucciones en el lazo.

El número máximo de lazos uno dentro del otro que BASIC puede ejecutar es 52.

Al tratar de ejecutar las declaraciones DO-UNTIL en el modo COMMAND generará un mensaje de error BAD SYNTAX.

Ejemplo

```
LAZO DO-UNTIL
> 10
> 20 I=I+1
> 30 PRINT1 I,
> 40 UNTIL I=5
> 50 PRINT1
> RUN
1 2 3 4 5

LAZOS UNO DENTRO DE OTRO DO-UNTIL
> 10 DO:I=I+1:DO:C=C+1
> 20 PRINT1 C, I*C
> 30 UNTIL C=3:C=0:PRINT1
> 40 UNTIL I=3
> RUN
1 1 2 2 3 3
1 2 2 4 3 6
1 3 2 6 3 9
```

DO-WHILE - Control de flujo

Función Ejecuta un lazo mientras prueba en la parte inferior del lazo que sea VERDAD

Sintaxis DO-WHILE *expresión de comparación*

Vea también DO UNTIL

Uso Las declaraciones DO-WHILE permiten que el usuario ejecute en varias ocasiones las líneas de programa entre las declaraciones DO y WHILE.

Cuando la *expresión de comparación* es igual a cero representa FALSO la ejecución del lazo termina; si no es así, la ejecución del lazo continúa.

El número máximo de lazos DO-WHILE uno dentro de otro que BASIC puede manejar es 52.

Al tratar de ejecutar las declaraciones DO-WHILE en el modo COMMAND se generará un mensaje de error BAD SYNTAX.

Ejemplo

```
LAZO D- WHILE
> 10  DO
> 20 I=I+1
> 30 PRINT1 I, "",
> 40 WHILE I<5
> 50 PRINT1
> RUN

1 2 3 4 5

LAZO DO-WHILE UNO DENTRO de OTRO
> 10 : I=I+1: : C=C+1
> 20 PRINT1 C, SPC(1), I*C,
> 30 UNTIL C=3: C=0: PRINT1
> 40 UNTIL  I<3
> RUN

1 1 2 2 3
3 1 2 2 4 3 6
1 3 2 6 3 9
```

DSR - Misceláneo

Función Obtener el estado de la linea de entrada del "handshake" de hardware

Sintaxis *var* = DSR *n*

Vea también DTR, SETPORT

Uso *n* especifica el puerto serial 1 o 2.

var vuelve el estado actual de la línea de CTS ya sea VERDAD (todos unos) o FALSO (cero).

Cuando el "handshake" de hardware es permitido por la declaración SETPORT, el protocolo del módem RTS/CTS controla automáticamente la declaración PRINT. Con el "handshake" de hardware deshabilitado, se puede poner en ejecución un "Handshake" como quiera por el usuario, con DSR.

Ejemplo La linea de entrada CTS en el puerto 1 (la clavija 5) está conectada con una línea de salida del "handshake" de hardware de un dispositivo externo. Por ejemplo, ésta podía ser la salida RTS o DTR de un dispositivo DTE.

Evita el desborde del buffer (buffer externo) de la entrada de los dispositivos externos.

```
1000 IF NOT DSR2 THEN RETURN
1010 REM El dispositivo en el puerto 2 está listo ( READY) para más datos
1020 PRINT2. . .
```

DTR - Misceláneo

Función Controla la salida de la línea del "handshake" de hardware

Sintaxis DTR *n* = *expresión*
variable = DTR *n*

Vea también DSR, SETPORT

Uso *n* especifica el puerto serial 1 o 2.

expresión hace OFF a la línea RTS para el puerto especificado cuando es igual a cero, sino se hace ON.

variable muestra el estado corriente de la línea RTS ya sea VERDAD o FALSO.

DTR2 se usa con el módem a bordo (cuando se tiene un módem) para forzarlo inmediatamente al estado de comando (incluso si estaba en línea). El control de programa por el usuario de esta clavija permite cambiar puertos sin colgar el teléfono o perder comunicación. DTR2 debe ser verdad para permitir el módem (DTR2=1).

Cuando el "handshake" de hardware es habilitado por la declaración SETPORT, el protocolo del módem RTS/CTS controla automáticamente la declaración PRINT. Con el "handshake" de hardware deshabilitado, se puede poner en ejecución un "Handshake" definido por el usuario con DTR.

Ejemplo 1 Módem en el puerto 2

```
> REM Apaga DTR para el puerto 2 para forzar al módem off line
> DTR2 = 0
> REM Enciende DTR para el puerto 2 para activar el módem
> DTR2 = 1
```

Ejemplo 2 La línea de salida RTS en el puerto 1 está conectada con una linea de entrada del "handshake" de hardware de un dispositivo externo. Por ejemplo, ésta podía ser la entrada de CTS o de DSR de un dispositivo DTE.

```
> STOP REM   El dispositivo externo para de transmitir
> DTR1 = 0
> REM permite transmitir al dispositivo
> DTR1 = 1
```

Ejemplo 3 Previene el flujo excesivo en el buffer (buffer)

```
> IF  INLEN1>127 THEN DTR1=0 ELSE DTR1=1
```

4

END - Control de flujo

Función Para la ejecución del programa

Sintaxis END

Vea también STOP

Uso La declaración END se usa para parar la ejecución de un programa BASIC.

El comando CONT, no funcionará para terminar la ejecución de programa si se usa la declaración END. BASIC terminará automáticamente el programa después de ejecutar la última línea de programa.

Ejemplo

```
REM hace funcionar el programa en memoria de datos en la energización
 AUTOSTART 1, 0, 9600
Mode = 1 RUN  (CLEAR)
Program = 0
Baud = 9600
READY

> 05 REM previene el acceso al programa y a datos a
> 06 REM personal desautorizado
>10 LOCKOUT = 1
>15 $(0)="secreto": REM  contraseña
>17 REM configure el timeout de INPUT  a 2 segundos
>20 SETINPUT 1,1,0,LEN($(0)),2000,2000
>30 INPUT, $(1)
>40 CHAR = 0
>50 DO
> 60 CHAR = CHAR + 1
> 70 IF  LEN($(0))=CHAR - 1 THEN LOCKOUT=0: END
> 80 WHILE MID$($(0), CHAR, 1)=UCASE$(MID$($(1), EL CHAR, 1))
> 90 REM Para programa si INPUT es igual a la contraseña
> 100 REM continúa

el programa principal comienza aquí.
. .
```

ERRCHK - Misceláneo

Función Verifica el error de un string o un bloque de la memoria de ABM

Sintaxis ERRCHK ($(*expr*), *n*) , K(*número de caracteres*), *tipo*
ERRCHK *dirección inicial*, *dirección final*, *tipo*
ERRCHK *dirección inicial*, K(*dirección final*), *tipo*

Vea también SYSTEM

4

Uso ERRCHK hace más rápida la construcción y recepción de mensajes para la mayoría de protocolos de comunicación ASCII.

La vcerificación de las fallas comienza con el e*n*ésimo carácter de la variable especificada de la string, $(*expr*). Si la expresión opcional *n* no se especifica ERRCHK comienza con el primer carácter en el string. El *número de caracteres* en el string en ERRCHK debe ser especificado.

Verifica el error de memoria del módulo BASIC especificando la *dirección inicial* y ya sea la *dirección final* o la *dirección final* para verificar errores.

El tamaño de bloque máximo que puede ser comprobado es 65535 bytes.

El *tipo* especifica el método 1, 2 o 3 de verificación de fallas.

1 LRC, control por redundancia longitudinal (XOR de bytes especificados)

2 CRC, control por redundancia cíclica (Polinomio = $X^{16}+X^{15}+X^{2}+1$)

3 Check Sum (la suma de bytes especificados/se divide por 256/el resto del número entero es el Check sum)

Los caracteres de verificación de error se almacenan en SYSTEM(5).

MSB = PICK(SYSTEM(5), H)

LSB = PICK(SYSTEM(5), L)

Ejemplo El check sum es un método simple de verificación de fallas que es usado por muchos dispositivos. El checksum de tipo ERRCHK es compatible con el protocolo de OPTOMUX.

```
100 REM Este  ejemplo enciende y apaga a los 16 canales de un
110 REM módulo de 16 salidas en la ranura 1
112 REM  Este programa es compatible con los
120 REM CPU 305-OPTO O CPU 305.
130 SETPORT 1,9600,N,8,1
140 SETPORT 2,9600,N,8,1,N,M
150 STRING 2551,254
160 SETINPUT 1,1,0,0,100,10
170 REM Envía el comando de reset al BRIDGE CPU.
180 $(1)="01A": GOSUB 2000
210 REM Hace y envía el comando de ENCENDER 16 salidas
```

```
220 $(0)="01K": GOSUB 2000
230 REM Construye y envía el string para apagar a 16 salidas.
240 $(0)="01l": GOSUB 2000
250 END
2000 REM Envía el comando OPTOMUX $(1) y obtiene la respuesta $(2)
2020 REM
2030 REM Calcula el checksum para la string de comando
2040 ERRCHK ($(1), 1), K(LEN($(1))), 3
2050 REM Agrega la checksum a la string de comando
2060 $(1)=$(1)+HEX$(DBY(25), 1)
2070 PRINT1 "COMMAND - >", $(0)
2080 PRINT2 ">";$(0);
2090 INPUT2 $(2); : PRINT1 "RECIBIDO - >", $(2)
2110 IF $(2)<>"A" THEN ERR=NOT(0) ELSE ERR=0
2120 RETURN
```

Ejemplo

El control por redundancia longitudinal LRC se usa en una gran cantidad de protocolos porque es más confiable que un simple CHECKSUM y más fácil de colocar en ejecución. El tipo LRC de ERRCHK es compatible con los protocolos *Direct*NET (lo mismo que HOSTLINK de TI, CCM2 de GE) y DYNAMIC RTU de TI.

```
110  REM  El ejemplo siguiente se usa para calcular el LRC para
120  REM un mensaje de HOSTLINK que escriba a los registros
130  REM 400-461. El programa requiere un cable del puerto 2
140 REM del módulo de BASIC del ASCII a la CPU 335.
150 REM
160 SETPORT 1,9600,N,8,1: SETPORT 2,9600,N,8,1
175STRING 2551,254 : DIM WT(100):
190 SETINPUT 1,1,0,0,100,10
200 Y=1: REM escribe un 1 a  cada registro
220 FOR I=1 TO  50: WT(I)=Y: NEXT I
230 REM envía un mensaje Enquire a la CPU 335
240 PRINT2 CHR$(78), CHR$(21H), CHR$(5);
250 INPUT2, $(1): REM Consigue Investiga El Ack.
290 $(0)=CHR$(1)+"01810041003200"+CHR$(17H): REM Construye
      encabezamiento
310 ERRCHK ($(0), 2), K(14), 1: REM calcula el encabezamiento LRC
330 ASC($(0), 17)=DBY(25): REM agrega LRC a la stringdel
      encabezamiento
350 PRINT2 USING(\17\), $(0); : REM envía el jefe fuera del puerto 2
360 INPUT2, $(1)
380 IF ASC($(1), 1)6 THEN  GOTO 190
```

```
390 REM Hace el string de datos de Write
400 $(0)=CHR$(2)
420 FOR POS=2 TO 51: ASC($(0), POS)=WT(POS-1):  NEXT POS
460 ASC($(0), POS)=3
480 ERRCHK ($(0), 2), K(50), 1: REM calcula LRC para escribir datos
490 ASC($(0), POS+1)=DBY(25)
500 PRINT2 USING(\53\), $(0); : INPUT2, $(1)
550 si ASC($(1), 1)=6 THEN Y=Y+1: REM Registro del topetón  si el resp es ack
560  GOTO 220
```


Avanzado

El control por redundancia cíclico es el más confiable de los tres métodos de verificación de fallas. Normalmente se usa el CRC-16 ya incluído en BASIC para comunicaciones. Esto se describe en el capítulo AVANZADO del manual de referencia de BASIC ampliado de FACTS Eng. El tipo CRC de ERRCHK es útil para verificar integridad de la memoria del programa y de los datos. ERRCHK también se usa para realizar un cálculo CRC-16 sobre una porción de un string después de que se haya hecho un INPUT.

Ejemplo

El ejemplo siguiente busca en un string un carácter de comienzo de un mensaje. ERRCHK se usa para calcular los caracteres CRC-16 del resto de string

```
300 REM construya un string de muestra con carácter inicial de texto (STX)
310 REM
320 $(0)="0123456789"+CHR$(2)+"0123456789"
330 REM
340 REM Encuentre donde se localiza el carácter STX
350 REM
360 POS=INSTR($(0), CHR$(2))+1
370 REM
380 REM Calcula el código de error CRC-16 de string $(0) que
390 REM comienza con el número después del carácter de STX
400 REM
410 ERRCHK ($(0), POS), K(10), 2
420  PRINT1 "STRING CON CRC - >", MID$($(0), POS, 10);
430 PRINT1 CHR$(DBY(25)), CHR$(DBY(26))
```

EXP - Operador matemático

Función Eleva el número "e" (2,7182818) a la potencia de la expresión.

Sintaxis EXP(*expresión*)

Uso Eleva el número "e" (2.7182818) a la potencia de la *expresión*.

Ejemplo

```
PRINT EXP(1)
2.7182818


PRINT EXP(LOG(2))
2
```

FOR-TO STEP-NEXT - Control de flujo

Función Ejecuta un lazo con índice ascendente o descendente con incremento automático

Sintaxis FOR *índice* = *índice inicial* TO *índice final* STEP *incremento*
NEXT *índice*

Uso A diferencia de muchos BASICS, las declaraciones FOR-TO-STEP NEXT se pueden ejecutar en tanto el modo RUN y COMMAND.

Estas declaraciones permiten que el usuario ejecute las líneas de programa entre las declaraciones FOR y NEXT por un número especificado de veces. Cuando se ejecuta la declaración, se asigna el valor de *índice inicial* a la variable *índice*. Cuando se ejecuta la declaración NEXT se agrega el valor *incremento* al *índice*. Luego *índice* es comparado con el valor de *índice final* . Si el *incremento* es positivo y el *índice* es menor o igual al *índice final*, entonces se transfiere el control a la declaración que sigue la declaración FOR. El *índice* también continuará incrementando si el *incremento* es negativo y el índice es mayor o igual al *índice final*.

La declaración STEP es opcional y si no se declara, el valor del *incremento* es 1.

La variable *índice* en la declaración NEXT es opcional y si falta, se asume ser la variable *índice* usada en la última declaración FOR.

Puede ser ejecutado un máximo de 9 lazos FOR NEXT, uno dentro de otro.

Ejemplo

```
> 10 FOR I=-3 TO 3
> 20 PRINT1 I, "",: NEXT : PRINT1
> RUN

-3 -2 -1 0 1 2 3
READY
>

> 10 FOR I=3 TO -3 STEP 2
> 20 PRINT1 I, "",: NEXT I: PRINT1
> RUN

3  1 -1 -3
READY
>
```

Muestre una región de la memoria del modo COMMAND

> FOR I=32768 TO 32768+5: PHO. CBY(I): NEXT
30H FFH FFH EEH 7FH FFH

Muestre el número decimal representado por la novena hasta la duodécima posición de bit de un número binario.

> FOR I=9 TO 12: P.2I, "",: NEXT**
512 1024 2048 4096

GO_PROGRAM o GOPRM - Control de flujo

Función Comenzar la ejecución de un programa con un número especificado

Sintaxis GO_PROGRAM, *número de programa*, *número de línea*

Vea también GOSUB, GOTO

Uso El *número de programa* identifica el programa almacenado para comenzar a ejecutarse y debe estar en el rango 0-255. GOPRM 0 especifica el programa en la memoria de datos y GOPRM 1 especifica el primer programa en el archivo de la memoria del programa. Si el *número de programa* de GOPRM especifica un número mayor que la cantidad de programas almacenados en la memoria de programa, entonces la declaración se ignora. Si el modo de reset del AUTOSTART es 2 o si opcionalmente se especifica el *número de línea* entonces todas las variables y strings se conservan después de una declaración de GOPRM.

GOPRM se podría usar para dividir una tarea de programación grande en programas más pequeños separados. Las ventajas de esta forma de programación son:

1) Programas más pequeños se ejecutarán más rápidamente (menos líneas a explorar).

2) Programas más pequeños se transfieren más rápidamente (modificaciones rápidas).

3) Programas más pequeños son más fáciles de documentar y de mantener.

4) Las variables del programa pueden ser locales o compartidas (globales).

5) Algunos de los programas más pequeños se podían usar en varias aplicaciones.

Ejemplo

```
02 REM  Programa principal en memoria de datos, PROGRAM = 0
04 REM El programa REG entra valores y configura registros
10 REG = 3
20 ALRM = 5:  REM  El programa ALRM muestra alarmas
22 REM El programa PRO muestra parámetros de proceso y
24 REM  valores corrientes de valores predefinidos
30 PRO = 2
. . .
1000 IF SETUP THEN GO_PROGRAM REG
. . .
2000 IF ALARM THEN GO_PROGRAM ALRM
. . .
3000 IF DISPLAY THEN GO_PROGRAM PRO
```

Ejemplo

```
GO_PROGRAM tiene acceso a "subrutinas" en otros programas
PRM 0
READY
> LIST
```

```
1000 REM Muestra "subrutinas" con GO_PROGRAM
1010 REM
1020 REM Pantallas de ayuda del usuario se almacenan en un
        programa archivado.
1030 REM Esto reduce el tamaño del programa principal.
1040 REM Se simplifica la mantención de ambos programas .
1050 REM
1060 REM inicializa los nombres de programa a la localización de
        programa
1070 REM
1080 HELP_PROG=3
1090 MAIN_PROG=0
1100 REM
1110 REM Inicializa el número de líneas de la "subrutina" de ayuda
1120 REM
1130 SETUP_HELP=2000
1140 DEBUG_HELP=4000
1150 CAL_HELP=6000
1160 REM
1170 REM El programa principal comienza aquí
1180 REM
1190 RESUME=SYSTEM(8): GO_PROGRAM HELP_PROG, SETUP_HELP
1200 PRINT2 "Completada la ayuda de configuración”
1210 RESUME=SYSTEM(8): GO_PROGRAM HELP_PROG, DEBUG_HELP
1220 PRINT2 "Ayuda completada de eliminar errores(DEBUG)"
1230 RESUME=SYSTEM(8): GO_PROGRAM HELP_PROG, CAL_HELP
1240 PRINT2 "Ayuda completada de calibración"
1250 END
PRM 3
READY
> LIST
2000 PRINT2 "Comienza Ayuda de configuración"
3999 GO_PROGRAM MAIN_PROG, RESUME
4000 PRINT2 "Comienza ayuda de DEBUG"
5999 GO_PROGRAM MAIN_PROG, RESUME
6000 PRINT2 "Comienza ayuda de calibración"
7999 GO_PROGRAM MAIN_PROG, RESUME
```

```
PRM 0
READY
> RUN

Comienza Ayuda de configuración
Completada la ayuda de configuración
Comienza ayuda de DEBUG
Ayuda completada de eliminar errores(DEBUG)
Comienza ayuda de calibración
Ayuda completada de calibración
```

GOSUB - Control de flujo

Función Ejecuta una subrutina

Sintaxis GOSUB *número de línea*

Vea también GO_PROGRAM, GOTO, RETURN

Uso GOSUB hace que BASIC transfiera control directamente a la línea de programa especificada por el *número de línea*. Cuando se encuentra la declaración RETURN en la subrutina, BASIC retorna el control del programa a la declaración inmediatamente después de GOSUB.

Ejemplo 1

```
SUBRUTINA
> 10 FOR I=1 TO  5
> 20 GOSUB 50
> 30 NEXT I
> 40 END
> 50 PRINT1 I, SPC(1),
> 60 RETURN
> RUN

1 2 3 4 5

READY
>
```

Ejemplo 2

```
UNA SUBRUTINA  DENTRO DE OTRA
> 10 FOR I=1 TO  5: GOSUB 50
> 20 I NEXT: END
> 30 A=I*I
> 40 RETURN
> 50 GOSUB 30: PRINT1 I, SPC(1), A, SPC(1)
> 60 RETURN
> RUN

1 1 2 4 3 9 4 16 5 25

READY
>
```

Ejemplo 3 **Terminación prematura de una subrutina sin CLEAR S**

```
> 10 GOSUB 20
> 20 I=I+1: IF I=100 THEN END
> 30 GOTO 10
> RUN

ERROR: CONTROL STACK OVERFLOW IN LINE 20

READY
> P.I
52
```

Ejemplo 4 **Salida prematura de una subrutina usando CLEAR S**

```
> 10 GOSUB 20
> 20 I=I+1: IF I=100 THEN END
> 30 CLEAR S: GOTO 10
> RUN

READY
> P.I
100
```

GOTO - Control de flujo

Función Transfiere la ejecución a la línea de programa especificada

Sintaxis GOTO *número de línea*

Vea también GO_PROGRAM, GOSUB

Uso La declaración GOTO hará que BASIC transfiera el control directamente a la línea de programa especificada por el *número de línea*. Si no existe un *número de línea*, será generado el mensaje, ERROR: INVALID LINE NUMBER.

Si se ejecuta la declaración GOTO en el modo COMMAND, BASIC no realiza el equivalente a la declaración CLEAR. En vez de eso, el control se transfiere a la línea de programa especificada con los valores sin cambio de todas las variables y del estado de interrupciones.

Si se ejecuta GOTO en el modo COMMAND después de que se haya corregido una línea, todas las variables se configuran a cero y todas las interrupciones son desactivadas.

Ejemplo

```
10 DEBUG=NOT(0)
20 IF NOT(DEBUG) THEN  GOTO 100
30 PRINT1 "Permitido debug , teclee CONT para continuar"

40 STOP

100 . . .
```

HEX$ - Operador de string

Función Convierte un número entero en su equivalente hexadecimal de string ASCII

Sintaxis *variable string* = HEX$(*expresión*, *1*)

Vea también OCTHEX$, PH0., PH1., STR$

Uso La *expresión* puede estar en el rango de 0 a 65535. El *1* es opcional y si incluído causa que HEX$ suprima ceros a la izquierda.

4

Ejemplo

```
> PRINT1 HEX$(10)
A

> PRINT1 HEX$(65535)
FFFF

> P. HEX$(800)
0320

> P. HEX$(10)
000A
```

HEX$ con los ceros a la izquierda suprimidos.

```
> P. HEX$(800,1)
320

> P. HEX$(10,1)
A
```

Relacionado Para convertir un string ASCII hexadecimal en un número entero, agregue un "0" al principio del string y de un "H" al final del string y use VAL.

Equivalente decimal de un string hexadecimal:

```
P. VAL("0"+"FFFF"+"H")
65535
```

IDLE - Interrupción

Función Suspende la ejecución de un programa hasta una interrupción

Sintaxis IDLE

Vea también ONPORT, ONTIME, RETI

Uso Las declaraciones IDLE fuerzan a BASIC a parar la ejecución de un programa hasta que se genera la interrupción especificada ya sea con una declaración ONTIME u ONPORT. Una vez que la interrupción ocurre, la rutina de interrupción es ejecutada y la ejecución del programa continúa con la declaración inmediatamente después de IDLE.

Observe que si BASIC incorpora una rutina de interrupción IDLE y el usuario ejecuta una declaración CLEAR I en la rutina de interrupción, el usuario debe volver a permitir la interrupción antes de salir de la rutina. Si esto no se hace entonces BASIC va a quedar parado hasta hacer un reset.

En caso de necesidad, IDLE se puede usar para disminuir el tiempo de respuesta de una interrupción.

IF -THEN- ELSE- Control de flujo

Función Ejecución condicional de declaraciones

Sintaxis IF *expresión de comparación* THEN *declaracion(es)* ELSE *declaración*

Uso si la *expresión de comparación* no es igual a cero (VERDAD), se ejecutan las *declaracion(es)* que siguen a THEN. Si la *expresión de comparación* es cero (FALSO) entonces se ejecuta la *declaración* que sigue a ELSE. Si se omite, la ejecución continúa con la línea de programa siguiente.

Se pueden ejecutar *declaracion(es)* múltiples separadas por dos puntos (:) después de THEN (si es VERDAD) o después de ELSE (si es FALSO).

4

Ejemplo

```
> 10 INPUT1 A
> 20 IF A<=2 PRINT1 "FIN": GOTO 30 ELSE GOTO 22
> 22 PRINT1 A/2, SPC(2),: A=A/2: GOTO 20
> 30 END
> RUN
?8
4 2 FIN
```

La palabra clave GOTO es opcional cuando es usada inmediatamente después de THEN o ELSE.

```
> 10 IF B*B>C THEN GOTO 50 ELSE GOTO 100
```

- O ALTERNATIVAMENTE -

```
> 10 IF B*B > C THEN 50 ELSE 100
```

La palabra clave THEN se puede substituir por cualquier declaración BASIC válida. Los ejemplos siguientes entregan el mismo resultado.

```
> 10 IF I=2 THEN 50 ELSE PRINT1 I

> 10 IF I=2 GOTO 50 ELSE PRINT1 I

> 10 IF I=2 THEN GOTO 50 ELSE PRINT1 I
```

4

INKEY$ - Operador de string

Función Entra un solo carácter sin hacer eco desde el buffer del puerto de entrada.

Sintaxis *variable de string* = INKEY$ *puerto*

Vea también INPUT

Uso INKEY$ quita el primer carácter en el buffer (buffer) de la entrada especificada por el puerto y lo asigna a *variable de string*.

INKEY$ vuelve un **retorno de carro** (ASCII 13) si el buffer de entrada está vacío (INLEN *puerto* = 0). Para distinguir entre un buffer vacío y un carácter de **retorno de carro** real, simplemente asegúrese que hay caracteres esperando en el buffer de entrada antes de ejecutar INKEY$.

INLEN *puerto* se puede usar para verificar si hay caracteres esperando en el buffer.

Ejemplo

```
1000  REM  Ejemplo de INKEY$
1010 REM El TXD del puerto 1vuelve a RXD
1015 REM
1020 SETPORT 1,9600,N,8,1,S,M
1022 REM
1025 REM Agrega una coma en el final de la declaración PRINT para
1026 REM suprimir el CR LF
1027 REM CR LF  no suprimido en este ejemplo
1028 REM
1030 PRINT1 CHR$(0),CHR$(13), "a",CHR$(13),CHR$(0), "b"
1040 PRINT2 INLEN1, "caracteres en buffer de la entrada"
1050 FOR I=1 TO INLEN1
1060 $(I)=INKEY$1
1070 IF ASC($(I), 1)=0 THEN PRINT2 "Null=", I:GOTO 1090
1080 IF ASC($(I), 1)=13 THEN PRINT2 "CR=", I:GOTO 1090
1082 IF ASC($(I), 1)=10 THEN PRINT2 "LF=", I:GOTO 1090
1085 PRINT2 $(I), "=",I
1090 NEXT
PRM 0
READY
> RUN


8 characters in input buffer
Null = 1
CR=2
a=3
```

```
CR=4
Null=5
b=6
CR=7
LF=8
PRM 0
READY
>
```

INLEN - Entradas - salidas

Función La función INLEN retorna la cantidad de caracteres que esperan en el buffer de entrada de una declaración INLEN. INLEN b*orra el buffer especificado de entrada.*

Sintaxis *conteo de caracteres* = INLEN *número de puerto*
INLEN *número de puerto* = 0

Vea también INPLEN, ENTRADA

Uso El *número de puerto* identifica el puerto de comunicación serial 1 o 2.

El *conteo de caracteres* es una variable que contiene el número de caracteres en el buffer especificado de entrada del puerto de comunicación. Si se llena el buffer de 255 caracteres, todos los caracteres adicionales se ignoran a excepción de *CONTROL-C* y de XON/XOFF (*CONTROL-Q/CONTROL-S*) y el *conteo de carácteres* continuará devolviendo 255.

Configurando el *número de puerto* de INLEN a cero borra el buffer de la entrada.

Ejemplo

```
10 REM Espera  por 10 caracteres en la entrada
11 REM  buffer
20 IF INLEN < 10 THEN GOTO 20
30 INPUT2, $(0)
40 IF INSTR($(0), "RA1")=1 THEN GOTO 100
50 REM Transmisión no para esta dirección remota
60 INLEN2 = 0: REM  Limpia el buffer de entrada
70 GOTO 20
100 REM resto de proceso del buffer de la entrada
. . .
```

INPLEN - Entradas - salidas

Función Muestra la cantidad de caracteres entrados con INPUT

Sintaxis *conteo de caracteres* = INPLEN

Vea también INLEN, INPUT

Uso INPLEN devuelve la cantidad de caracteres recibidos por la última declaración de INPUT ejecutada. INPLEN es levemente más rápido que la declaración LEN(~1 milisegundo). INPLEN es útil cuando se entran strings ASCII de 8 bits con INPUT o de datos binarios que puedan incluir un ASCII 13 (LEN parará de contar caracteres cuando encuentra un carácter **retorno de carro**, ASCII 13).

Ejemplo

```
10 STRING 2551.254: REM 10, 254 strings
20 INPUT $(0)
30 IF INPLEN > 2 THEN PRINT1 INPLEN

> RUN

?STRING LENGTH=?
17

10 INPUT A
20 PRINT1 INPLEN: REM Imprima longitud del último INPUT
> RUN

?135.6
5
```

4

INPUT - Entradas - salidas

Función Carga variables con datos desde el puerto1 o 2

Sintaxis INPUTn *aviso de string, variable, variable...*

Vea también INKEY$, SETINPUT, SETPORT

Uso *n* especifica el número de puerto que contiene los datos o los caracteres para la lista de *variables*. Si se va a colocar más que una variable numérica adentro de una declaración INPUT, entonces cada número se debe separar por una coma (,). Por defecto, un carácter de **retorno de carro** señala el fin de una lista de datos de entradas numéricas y de string.

El *aviso de string* es una constante opcional de string. Si se omite el *aviso de string*, será mostrado un signo de interrogación (?) en la pantalla para los datos. Si se pone una coma antes de la primera *variable* que sigue un INPUT entonces no será enviado el aviso del signo de interrogación.

La operación de INPUT es controlada por la declaración de SETINPUT. INPUT y SETINPUT pueden realizar más funciones que las declaraciones INPUT$, INPUT#, y LÍNE INPUT encontrada en otros programas BASIC. La capacidad de entrar sin eco es única al BASIC ampliado de FACTS Eng., la capacidad para redefinir el carácter de terminación de INPUT(por ejemplo = en vez de CR) y para controlar el tiempo que INPUT esperará datos (vea SETINPUT).

Una lista de *variables* de string funciona igual que las declaraciones múltiples de INPUT, sin embargo, INPLEN solamente devolverá el número de caracteres entrados en la última *variable* de string.

Si se usa una lista de *variables* numéricas entonces cada número entrado se debe separar por una coma (,). Se debe entrar un **retorno de carro** para señalar el fin de la listas de variables numéricas. Este método de entrada de datos **no se recomienda** para la mayoría de las aplicaciones.

Ejemplo

```
> 10 INPUT NUM, D1: REM Entre un RETURN después de cada  #
> 20 INPUT, D: REM Aviso cuando no hay coma ?
> 30 SETINPUT 1: REM permite no-eco
> 40 INPUT, $(0)
> 30 PRINT NUM, SPC(1), D1, SPC(1), D, SPC(2), $(0)
> RUN

?10, 30
5
10 30 5

READY
>
```

Manejo de los errores de INPUT

Si no se entran datos con INPUT para cada variable numérica de una lista de entrada entonces las variables en la lista no se cambian.

Si se entra un carácter alfanumérico para una *variable* numérica entonces aparece el mensaje TRY AGAIN (TRATE OTRA VEZ).

Cuando se entran más datos numéricos que variables en la lista de INPUT, aparece el mensaje EXTRA IGNORED (ignorados) y todos los datos son ignorados hasta el próximo carácter de terminación de INPUT(generalmente un **retorno de carro**).

Debido a las limitaciones antedichas es casi siempre mejor entrar datos numéricos en un string y después convertir el string en un número.

Ejemplo

```
REM Puede haber un loop sin fin si los datos no se entran correctamente
10 INPUT1 “Entre la hora (hora, minuto, seg)", HORA, MINUTO, SEG
20 PRINT1 "EL TIEMPO ACTUAL ES",HR, ":",MIN, ":",SEG
RUN

Entre la hora (hora, minuto, seg)
10 30 47

INPUT must be a number, TRY AGAIN
Entre la hora (hora, minuto, sec)
10, 30, 47
EL TIEMPO ACTUAL ES 10:30:47


REM Un método mejor es siempre entrar datos en un string
10 TRYS=0
15 INPUT1 "Entre la hora (Hora, minuto, segundo)", $(0)
20 TRYS = TRYS + 1
25 HORA = VAL($(0))
30 IF HORA > = 0.AND.HORA<=23 THEN GOTO 50
35 PRINT "La hora debe ser < = 23"
40 IF TRYS = 3 THEN GOTO 100 : REM salta si el operador erró
45 GOTO 15
50POS_MIN = INSTR($(0), ":")+1
55 $(1) = MID$($(0),POS_MIN)
60 MINUTO = VAL($(1))
70 POS_SEG = INSTR($(1), ":")+1
80 SEG = VAL(MID$($(0), POS_SEG))
90 PRINT1 "El tiempo actual es",HR, ":",MINUTO, ":",SEG
95 TIME$=STR$(HR)+":"+STR$(MIN)":"+STR$(SEG)
100. . .
```

Especial Cuando hay más datos numéricos presentes que variables en la lista de INPUT entonces se genera el mensaje EXTRA IGNORED y todos los datos se ignoran hasta la próximo carácter de terminación INPUT, generalmente **un retorno de carro**).

Ejemplo

```
10 INPUT A, B
>
RUN
?234, 42, 10
EXTRA IGNORED
```

Entrada no estándar de caracteres ASCII

Los caracteres de control (ASCII 0 - 31) por defecto hacen eco pero no se cargan en variables. Para entrar con INPUT caracteres de control use SETINPUT para configurar *no-edit* ON.

Para entrar códigos especiales de 8 bits que no son una parte del uso estándar del conjunto de caracteres ASCII (ASCII 128 a 255) use SETPORT para seleccionar 8 bits de datos.

Ejemplo

```
REM Hace que el eco sea ON y el carácter de control INPUT ON
10 NO_ED = 1
20 NO_ECHO = 0
30 SETINPUT NO_ECHO, NO_ED
40 GOSUB 100
45 REM Deshabilita entrada de caracter de control, permite modificar
46 REM entrada
50 SETINPUT NO_ECHO, 0
60 PRINT1: GOSUB 100
70 END
100 INPUT "Entre < Ctrl-G >, Back_space, 234", $(0)
110 PRINT1 "Longitud del string entrado =", INPLEN
120 PRINT1 "Primer carácter del string es ",
121 PRINT1 LEFT$($(0), 1)
130 RETURN
RUN

Entre < Ctrl-G >, Back_space, 1234
1234
Longitud del string entrado = 6
Primer carácter del string es (el terminal suena debido al carácter BELL)
Entre < Ctrl-G >, Back_space, 1234 1234
Longitud del string entrado = 4
Primer carácter del string es 1
```

Caso especial de una entrada de carácter de control

Existe un caso especial de entrada de carácter de control cuando los caracteres de control ASCII NULL < Ctrl-@ > (, ASCII = 0) y < Ctrl-M > (CR, ASCII = 13) representan datos. Estos caracteres son usados como delimitadores por toda las declaraciones que manejan strings excepto ASC. Por lo tanto, sólo puede ser usada la función ASC para procesar strings conteniendo los caracteres NULL y del CR como datos.

Ejemplo

```
10 NO_ED = 1
11 REM No permite corrección (entrar caracteres de control)
20 SETINPUT 0, NO_ED, 0, 5, 10000, 2000
30 PRINT1 "Entre ”,CHR$(34),”12 < Ctrl-M > < Ctrl-@ >",
31 PRINT1 "3", CHR$(34)
40 PRINT1 "Tiene 5 segundos para entrar el primer carácter"
50 INPUT1, $(0)
60 PRINT1 "Longitud de la entrada =", INPLEN
70 PRINT1 "Declaración LEN dice la longitud de string =",
71 PRINT1 LEN($(0))
75 PRINT1 "Valores ASCII para todos los caracteres INPUT: ",
80 FOR POS = 1 TO INPLEN
90 PRINT ASC($(0), POS), SPC(2),
100 NEXT POS

> RUN
Entre "12 < Ctrl-M > < Ctrl-@ > 3"
Tiene 5 segundos para entrar el primer carácter
3
Declaración LEN dice la longitud de string =
Longitud de la entrada = 5
Declaración LEN dice la longitud de string = 2
Valores ASCII para todos los caracteres INPUT: 49 50 13 0 51
```

4

INSTR - Operador de string

Función INSTR busca en un string por un string patrón

Sintaxis *posición* = INSTR (*string base, expresión patrón de string*)

Uso INSTR vuelve la *posición* del *patrón de string* en el *string base*. Si no se encuentra la *expresión patrón de string* en la búsqueda en la *string base*, entonces la *posición* será 0. Si cualquier string tiene una longitud de 0 entonces INSTR vuelve un 0. Ambos strings pueden ser expresiones de string.

Ejemplo

```
10 STRING 2551,254
20 INPUT "Entre expresión de string a buscar", $(0)
30 $(1) = "Contraseña"
40 POS = INSTR($(0), $(1))
50 IF POS = 0 THEN PRINT1 "ACCESO NEGADO": END
60 PRINT1 "contraseña está correcta"

> RUN
Entre expresión de string a buscar  CONTRASEÑA
ACCESO NEGADO
READY
> RUN
Entre expresión de string a buscar Contraseña
Contraseña está correcta

10 STRING2551,254
20 $(0)="LunMartMiercJuevViernSabDom"
30 INPUT "Entre el día de la semana? ", $(1)
40 IF INSTR($(0), $(1))=0 THEN GOTO 30
50 PRINT1  "Este día es la posición ", INSTR($(0), $(1))

> RUN
Entre el día de la semana? Mierc
Este día es la posición 8

READY
> P. INSTR($(0), "Mart")
 3
```

INT - Operador matemático

Función Calcula la porción entera del número entero de la expresión.

Sintaxis INT(*expresión*)

Uso Calcula la porción entera del número entero de la *expresión*.

Ejemplo

```
PRINT INT(3.7)
3

PRINT INT(100.876)
100
```

4

LCASE$ - Operador de string

Función LCASE$ devuelve una string que consiste solamente de caracteres minúsculos

Sintaxis *variable de string* = LCASE$(*expresión de string*)

Vea también UCASE$

Uso LCASE$ vuelve una string igual a la *expresión de string* excepto que todos los caracteres alfabéticos mayúsculos en la *expresión de string* se convierten en minúsculas.

Ejemplo

```
> 10 PRINT1 "Imprima informe sumario del año hasta la fecha?"
> 11 INPUT1, "(y/n)", $(0)
> 20 IF LCASE$($(0))="y" THEN GOTO 100
> 30 PRINT1 LCASE$("Impresión cancelada!")
> 40 END
100 REM Imprime informe sumario del año hasta la fecha
. . .

> RUN
Imprima informe sumario del año hasta la fecha? (y/n) ¡ N
Impresión cancelada!

READY
>
```

LEFT$ - Operador de string

Función LEFT$ vuelve una cadena de n caracteres comenzando con el primer carácter

Sintaxis *variable de string* = LEFT$(*expresión de string*, *n*)

Vea también MID$, REVERSE$, RIGHT$

Uso *n* es una expresión y especifica la cantidad de caracteres de la *expresión de string* que se asignará a la *variable de string*. *n* debe estar en el rango 0 a 254.

LEFT$ vuelve una string que consiste en una cadena de caracteres desde el primer carácter hasta el enésimo de la *expresión de string*.

Si *n* es mayor o igual que la longitud de la *expresión de string* entonces toda la *expresión de string* se asigna a la variable de string.

Si *n* es 0 entonces LEFT$ devuelve un string sin caracteres.

Ejemplo

```
> PRINT1 LEFT$("PUEDE HACERLO, 3); "HACERLO"
NO HACERLO

READY
>
```

4

4

LEN - Operador de string

Función LEN devuelve la cantidad de caracteres en una string

Sintaxis LEN (*expresión de string*)

Uso LEN devuelve la cantidad de caracteres en la *expresión de string*, 0 a 254.

Ejemplo

```
10 STRING 2551,254: REM asigna 10, longitud máxima
20 INPUT "Entre por favor un string", $(0)
30 PRINT1 "La longitud de string es", LEN($(0)

> RUN
Entre por favor un string   OK, Un string
La longitud de string es 13

READY
> $(0)="ABCDEFGHIJK"
> P. LEN(LEFT$($(0), INSTR($(0), "E")))
5

READY
>
```

LET - Misceláneo

Función LET asigna el valor de una expresión a una variable

Sintaxis LET *variable* = *expresión*

Uso *expresión* es una expresión numérica o un string (cadena de caracteres) cuyo valor se asigna a una *variable*. La palabra clave LET es opcional.

Ejemplo

```
10 STRING 2551,254
15 A = 123.4 * 10
20 $(0) = "UNO"
30 $(1) = "DOS"
40 $(2) = $(0) + $(1) + "TRES"
50 $(3) = CHR$(38) + "CUATRO"
60 PRINT1 CHR$(34)+$(2)+$(3)+CHR$(34)+ "=",A

> RUN
"UNODOSTRES&CUATRO" = 1234
```

La declaración siguiente es también válida.

```
60 PRINT1 CHR$(34), $(2), $(3), CHR$(34), "=",A

10 STRING 2551,254
20 FOR CONTEO = 1 TO  50
30 $(0) = $(0) + "*"
40 NEXT CONTEO
50 PRINT1 $(0)
60 PRINT1: PRINT1 SPC(21), "ENCABEZAMIENTO"
70 PRINT1: PRINT1 $(0)
> RUN
**************************************************
                     ENCABEZAMIENTO
**************************************************
```

4

LOAD@ o LD@ Operador avanzado

Función Recupera un número de coma flotante de seis bytes desde la memoria

Sintaxis LOAD@ *dirección*

Vea también BYTE, STORE@, WORD

Uso LOAD @ permite que el usuario recupere los números de coma flotante almacenados en memoria de datos con la declaración STORE@. La *dirección* es la dirección de memoria más alta en donde se almacena el número.

La ejecución de la declaración LD@ pone un número en el "stack" del argumento del cual BASIC puede asignarlo a una variable con la declaración POP.

Puesto que un número de coma flotante requiere seis bytes de almacenaje, la declaración ST@ 32767 archivaría el último número empujado (PUSHed) al "stack" en las direcciones 32767, 32766, 32765, 32764, 32763, y 32762.

Ya que BASIC almacena strings y variables sin dimensión en memoria desde MTOP para abajo, el usuario debe configurar una porción de memoria libre que se usará por las declaraciones ST @ y LD @.

Ejemplo Asigna una región protegida de memoria para almacenaje de variables

```
> MTOP=28000: REM configura y almacena el nuevo valor MTOP
> AUTOSTART modo, programa, baud, 28000

PUSH 1234.56 Número del lugar en que se almacenará en "stack"

> ST@ 28000+7 Almacena número en memoria de datos sobre MTOP

> LD@ 28007 Recupera (carga) el número almacenado

> POP  NUM       Asigna el número recuperado a una variable

> PRINT NUM
1234.56

>05 REM almacena en memoria de datos números de coma flotante
> 10 DIM D(3) : D(1) = 907.701
> 20 D(2) = 3256
> 30 D(3) = 39.2E+9
> 40 INDICE  = 1
> 50 FOR MEM = 28007 TO 28007+2*6 STEP 6
> 51 REM MEM apunta al valor
> 60 PUSH D(INDICE)
> 70 ST@ MEM: REM almacena el valor
> 80 INDICE = INDICE + 1
> 90 NEXT MEM
```

```
> RUN  los tres valores ahora se almacenan en memoria

> 105 REM Carga de nuevo los números almacenados arriba
> 110 FOR MEM = 28007 TO STEP 28007+2*6 STEP 6
> 115 REM MEM apunta a los números
> 120 LD@ MEM
> 130 POP NUM
> 140 PRINT1 NUM
> 150 NEXT MEM
> RUN

907.701
3256
3.925 E+10
```

4

LOCKOUT - Control de flujo

Función Fuerza la ejecución de una programa

Sintaxis LOCKOUT = *verdad o falso*

Vea también BREAK

Uso *verdad o falso* es una expresión que cuando es igual a 0 deshabilita LOCKOUT. Si *verdad o falso* es diferente a cero entonces BASIC no volverá al modo de comando. Si se incorpora un < *CTRL-C* >, se ejecuta una declaración END o STOP, o un error es generado y luego BASIC recomenzará el módulo basado en los parámetros actualmente almacenados en AUTOSTART.

La declaración LOCKOUT es usada para suministrar seguridad de programa y variable de datos previniendo el acceso a personal desautorizado. Si se habilita el LOCKOUT, el modo de comando puede ser obtenido solamente removiendo el módulo de la base y moviendo el puente CLEAR ALL en la tarjeta de circuito impreso a la posición que deshabilita la función de AUTOSTART y borra toda la memoria de datos después de un reset (vea el manual de usuario específico del módulo). LOCKOUT podría también habilitar o deshabilitar el programa con una contraseña según lo mostrado en el ejemplo para la declaración END.

La declaración LOCKOUT también se usa para recuperar con seguridad anomalías del programa BASIC y de condiciónes de entradas inesperadas o eventos externos.

Ejemplo

```
05 REM se recupera de una condición de error  no atrapada
10 LOCKOUT = NOT(0): REM Fuerza ejecución de programa
20 INPUT1 "Código a enviar al PLC" CODE
30 DUMMY = TRANSFER (CODE)
40 LOCKOUT = 0: REM Deshabilita LOCKOUT
> RUN
Código a enviar al PLC
?300

ERROR: BAD ARGUMENT - IN LINE 30
30 DUMMY = TRANSFER (CODE)
-------------------X
READY
Código a enviar al PLC
?255
```

LOF - Gerencia de memoria

Función Devuelve el tamaño del programa seleccionado

Sintaxis LOF

Uso LOF le dice al usuario cuántos bytes de memoria ocupa el programa actualmente seleccionado. LOF se puede usar en tanto en el modo RUN y el modo COMMAND.

LOF se puede usar para comparar el tamaño del programa que está siendo modificado con el espacio libre disponible en el archivo de almacenaje del programa.

LOF se puede usar para determinar el número de bytes de memoria RAM que esté disponible para el almacenaje de strings y variables numéricas.

LOF no define el número de bytes de memoria usados corrientemente para el almacenaje de strings y variables numéricas.

Ejemplo

```
> PRM 0
> P. 32767 - LOF - 1279
28345

READY
>
```



32767 = Parte superior de la memoria de datos (MTOP)
1279 = memoria de datos usada por el intérprete

LOG - Operador matemático

Función Calcula el logaritmo natural de la expresión (con base e)

Sintaxis LOG (*Expresión*)

Uso Calcula el logaritmo natural de la *expresión*. La *expresión* debe ser mayor de 0. Este cálculo se realiza con 7 dígitos significativos.

Ejemplo

```
PRINT LOG(12)
2.484906

PRINT LOG(EXP(1))
1
```

4

MID$ - Operador de string

Función MID$ devuelve una cadena de *m* caracteres comenzando con el e*n*ésimo carácter

Sintaxis *variable de string* = MID$(*expresión de string, n, m*)

Vea también LEFT$, REVERSE$, RIGHT$

Uso MID$ devuelve una string que comienza con el e*n*ésimo carácter de la *expresión de string*. *m* es una expresión y especifica el número de caracteres de la *expresión de string* que se asignará a la *variable de string*. *n* y *m* deben estar en el rango 0 a 254.

Si se omite *m* o hay menos que *m* caracteres a la derecha del enésimo carácter de la expresión de la secuencia, entonces todos los caracteres restantes de la *expresión de string*, se asignan a la *variable de string*.

Si *n* es 0 o mayor que la longitud de la *expresión de string*, MID$ devuelve el carácter **Null**.

Ejemplo

```
> 10 STRING 2551,254: REM asigna longitud de 10 máximo
> 20(0)="1JAN2FEB3MAR4APR5MAY6JUN7JUL8AUG9SEP10OCT11"
> 21 $(0)=$(0)+"NOV12DEC13"
> 30 MES = 10
> 40 START= INSTR($(0), STR$(MES))+1
> 50 STP = INSTR($(0), STR$(MES+1))
> 60 PRINT1 "EL MES ES",
> 61 PRINT1 MID$($(0), START, STP-START)

> RUN
El mes es OCT

READY
>
```

MTOP - Operador avanzado

Función Límitar la memoria disponible por el intérprete BASIC

Sintaxis *variable* = MTOP

MTOP = *dirección*

Uso Después de reset, BASIC asigna normalmente un valor a MTOP leyendo el valor almacenado al principio de la memoria del programa por AUTOSTART.

Vea el manual de usuario específico del módulo para el comportamiento de MTOP.

BASIC no usará ninguna memoria variable más allá de la dirección asignada a MTOP.

Si la dirección es mayor que la última dirección de memoria válida, entonces será generado un error MEMORY ALLOCATION.

Si MTOP se usa en un programa, debe ser la primera declaración en el programa porque BASIC almacena strings y variables sin dimensión desde MTOP para abajo.

Ejemplo

```
> PRINT MTOP
32767    (valor prefijado)

> MTOP=16383   (asigna un valor nuevo)

> PRINT MTOP
16383

REM almacena el valor nuevo de MTOP en la próxima energización
> AUTOSTART mode, program, baud, 16383
```

OCTHEX$ - Operador de string

Función Convierte un número octal (de base 8) en su equivalente string hexadecimal ASCII

Sintaxis *variable de string* = OCTHEX$(*expresión*, *1*)

Vea también HEX$, PH0., PH1.

Uso La *expresión* debe estar en el rango de 0 a 177777. El *1* es opcional y si es incluído hace que OCTHEX$ suprima ceros a la izquierda.

Ejemplo

```
PRINT1 OCTHEX$(10)
0008
PRINT1 OCTHEX$(177777)
FFFF
P. OCTHEX$(7777+1): REM Última dirección de memoria V de usuario
1000
OCTHEX$ con ceros a la izquierda suprimidos.
P. OCTHEX$(700,1)
1C0
P. OCTHEX$(10,1)
8
```

Relacionado Para convertir una string hexadecimal ASCII en un número decimal, agregue un "0" al principio del string y una "H" al fin del string y use VAL.

```
P. VAL("0"+OCTHEX$(7777+1)+"H")
4096
```

Avanzado Use este comando para convertir una dirección octal conocida de Memoria V en el equivalente hexadecimal. Esta declaración es útil para las tablas de look-up y otros tipos de accesos de memoria "calculados" del PLC.

Asuma que un operador suministra la dirección inicial de memoria V de una tabla de look-up. Este valor es 1400. La dirección hexadecimal equivalente es

```
STRADDR = VAL("0"+OCTHEX$(1400+1)+"H")
```

El valor del décimo elemento en la tabla de look-up especificada es

```
TBL(9) = S405_(STRADDR+9)
```

El décimo elemento en la tabla está en la dirección octal de memoria V1411

ON-GOSUB - Control de flujo

Función Llama la subrutina comenzando en uno de varios números de líneas posibles

Sintaxis ON *expresión* GOSUB *número de línea, número de línea*

Vea también ON GOTO

Uso La *expresión* selecciona el *número de línea* inicial para una llamada de subrutina. Si la *expresión* evalúa como cero entonces la ejecución continúa en la línea de programa especificada por el primer *número de línea* en la lista. Después de que se ejecute una declaración RETURN en la subrutina, la ejecución se reanuda con la declaración que sigue ON-GOSUB.

Si el valor de la *expresión* es mayor que o igual al número de *número de línea* en la lista, entonces será generado el mensaje de error BAD SYNTAX.

Ejemplo

```
10 IF (MODELO<0).OR.(MODELO>3) THEN GOSUB 100
20 ON MODELO GOSUB 1000, 2000, 3000, 4000
. . .
100 REM subrutina para entrar el número de modelo
150 RETURN
. . .
1000 REM Construya arreglos para la fabricación MODELO=0
1999 RETURN
. . .
2000 REM Construya arreglos para probar MODEL0=1
2999 RETURN
. . .
3000 REM Construya arreglos para supervisar MODEL0=2
3999 RETURN
. . .
4000 REM  Construya arreglos para construir el MODELO=3
4999 RETURN. . .
```

ON-GOTO - Control de flujo

Función Saltar a uno de varios números de línea posibles

Sintaxis ON *expresión* GOTO *número de línea, número de línea*

Vea también ON GOSUB

Uso La *expresión* selecciona el número de línea del programa donde la ejecución continuará, sin condición.

Si la *expresión* evalúa como cero entonces la ejecución continúa en la línea de programa especificada por el primer *número de línea* en la lista.

Si el valor de la *expresión* es mayor o igual al número del *número de línea* en la lista, entonces aparece un mensaje de error BAD SYNTAX.

Ejemplo

```
05 REM Muestre mensajes en una sola línea
10 FOR I = 1 TO  4
15 PRINT2 $(0): REM Borra lo que está en la pantalla y vuelve a la línea
20 ON I - 1 GOTO 100, 110, 120, 130
30 NEXT I
40 END

100 PRINT2 "* * * Atención ***"
105 DELAY 2: GOTO 30
110 PRINT2 "Ciclo automático de la Máquina comienza ahora"
115 DELAY  4: GOTO 30
120 PRINT2 "Proceso comienza en el modelo", $(1)
125 DELAY  3 + LEN($(1))/10: GOTO 30
130 PRINT2 "Presione el botón RESET para cancelar"
140 DELAY 3.5: GOTO 30
```

ONERR - Control de flujo

Función Especifica la línea de programa a que debe ir si ocurre un error aritmético

Sintaxis ONERR *número de línea*

Vea también SYSTEM

Uso Si ocurre un error aritmético después de que se ejecute la declaración de ONERR, BASIC pasará control al *número de línea* del programa especificado en la última declaración ONERR. La declaración ONERR atrapa solamente errores aritméticos.

El usuario puede examinar la posición de memoria de datos (BYTE) 257 (101H) en una rutina de gestión de errores para determinar que condición de error ocurrió.

Condición de Error	Código de Error
DIVISIÓN POR CERO	10
DESBORDAMIENTO(OVERFLOW)	20
DESBORDAMIENTO INFERIOR (UNDERFLOW)	30
MAL ARGUMENTO	40

Ejemplo

```
> 10 ONERR 100:I=4
> 20 PRINT1 100/I,
> 30 I=i-2
> 40 GOTO 20
> 100 IF BYTE(257)=10 THEN PRINT1 "ERROR DIVISIÓN POR CERO"
> 110 IF  BYTE(257)=20 THEN PRINT1 "OVERFLOW"
> 120 IF  BYTE(257)=30 THEN PRINT1 "UNDERFLOW"
> 130 IF  BYTE(257)=40 THEN PRINT1 "MAL ARGUMENTO"
> 140 END
> RUN
25
50
ERROR DIVISIÓN POR CERO
```

4

ONPORT - Interrupción

Función Especifica el número de línea para gerenciamiento de eventos del puerto serial

Sintaxis ONPORT *n, número de línea*

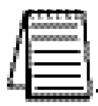
Vea también IDLE, RETI

Uso ONPORT habilita la interrupción del flujo normal del programa BASIC al recibir un carácter en el puerto serial especificado por *n*. El *número de línea* es la línea inicial de programa para la subrutina que maneja la interrupción de ONPORT. La declaración ONPORT permite ocurrir solamente una sola interrupción del programa BASIC.

Los eventos futuros en el puerto serial especificado no se atrapan (interrupción permitida) hasta que se ejecuta otra declaración ONPORT. Por lo tanto, otra declaración ONPORT sería incluída normalmente en la subrutina de la interrupción si debe continuar la interceptación de evento del puerto serial.

Un *número de línea* 0 deshabilita la interrupción especificada por ONPORT.

Una interrupción habilitada por ONPORT hace que la ejecución del programa continúe en el *número de línea* que sigue después de la terminación de la declaración corriente.



NOTA: *ONPORT no espera la terminación de las declaraciones DELAY o IDLE antes de pasar control a la rutina de la interrupción de ONPORT.*

Después de que se ejecute una declaración RETI en la subrutina que maneja la interrupción, la ejecución se reanuda con la declaración que sigue la última declaración ejecutada antes de que ocurrió la interrupción.

Ejemplo

```
10 REM Lazo principal del programa
20 Rem Aquí obtenemos el estado de la lógica del PLC.
30 Rem Si el estado de la lógica indica una falla cíclica entonces
40 Rem obtenemos el estado de entradas y salidas para determinar
50 REM  la causa y mostrarla (en el puerto 1).
60 REM Si no es así, muestre parámetros de proceso.
70 ONPORT2, 1000: REM atrapa la entrada del código de barras
. . .
500 10 GOTO: REM Fin del lazo principal del programa
. . .
1000 REM string de proceso de  datos del código de barras
. . .
1400 ONPORT2, 1000: Rem supervisa al lector de código de barras
1410 RETI
```

La declaración ONPORT se puede usar para más rapidez y conveniencia. En aplicaciones

menos sensitivas al tiempo de respuesta podría también verificar regularmente considerar si hay algunos caracteres esperando en el buffer de entrada usando la declaración INLEN (**IF INLEN2>0 THEN GOSUB ...**).

ONTIME - Interrupción

Función Interrupción basada en tiempo del flujo de programa normal

Sintaxis ONTIME *tiempo predefinido, número de línea*

Vea también IDLE, RETI, SYSTEM, TIME

Uso ONTIME permite la interrupción del flujo de programa normal BASIC cuando el valor de TIME es mayor o el igual al valor del *tiempo predefinido* . El *tiempo predefinido* puede ser cualquier valor entre 0,005 a 65535,995 segundos. El *número de línea* es la línea de programa inicial para la subrutina que maneja la interrupción ONTIME. RETI señala el final de la subrutina.

La declaración ONTIME permitirá solamente una sola ocurrencia de interrupción del programa BASIC. No ocurrirán futuras interrupciones TIME hasta que se ejecuta otra declaración ONTIME. Por lo tanto, otra declaración de ONTIME sería incluída normalmente en la subrutina de la interrupción.

Un *número de línea* 0 deshabilita la interrupción de ONTIME.

Una interrupción permitida por ONTIME causa que la ejecución del programa continúe en el *número de línea* especificado después de completar la declaración corriente. Después de que se ejecute una declaración RETI en la subrutina que maneja la interrupción, la ejecución se reanuda con la declaración que seguía la última declaración ejecutada antes de que ocurriera la interrupción.

Ejemplo

```
> 10 TIME=0
> 20 ONTIME 2000, 100
> 30 INPUT "UN NUMER0 "x
> 40 PRINT1 X, "", TIME
> 50 END
> 100 PRINT1 "INTERRUPCIÓN"
> 110 RETI
> RUN

UN NÚMERO
?10      (ESPERA POR LO MENOS 2 SEGUNDOS)
 INTERRUPCIÓN
10 3.945

> 10 TIME=0: DBY(71)=0: REM  Temporizador en cero
> 20 CLOCK 1: REM comienza el contador de tiempo
> 15 ONTIME 1, 100: REM permite la interrupción en la línea 100
> 30 REM nada en este ejemplo
> 40 UNTIL INFINITO
```

```
> END 50
> 100 PRINT1 "PROGRAMA INTERRUMPIDO PERIÓDICAMENTE"
110 REM La Procima inrterrupción  ocurrirá en 3 segundos después
> 115 IF TIME>65000 THEN TIME=TIME-65000
> 120 ONTIME TIME+3, 100
> 130 RETI
```

Prioridad de la interrupción - ONPORT y ONTIME

BASIC ampliado de FACTS Eng. establece una prioridad más alta para la interrupción de ONTIME que para las interrupciones ONPORT. Es decir, una interrupción de ONTIME puede interrumpir una interrupción de ONPORT. Esta prioridad fue establecida para poder lograr tareas basadas en tiempo crítico, tales como poder mantener un control de PID.

Para evitar que una interrupción de ONTIME ocurra durante un subrutina de interrupción de ONPORT, pare temporariamente el temporizador de software.

Ejemplo Prevención de interrupciones de ONTIME

```
10TIME = 0: DBY(71)=0: REM pone a cero el temporizador
100 ONPORT1, 1000: REM maneja la entrada del operador
110 ONTIME 2, 2000: Muestra TIME$/DATE$
200 REM lazo principal del programa REM
. . .
500 200 GOTO: REM Finaliza el lazo principal del programa

1000 ONTIME 0: REM deshabilita la interrupción de ONTIME
1010 REM  procesa la entrada del operador
 . . .
1400 IF INLEN1>0 THEN GOTO 1010: REM Vaya a 1010 si necesario
1410 ONPORT1, 1000:REM habilita el próximo ONPORT  interno.
1510 ONTIME 2, 2000: Rem Reactiva el ONTIME interno.
1520 RETI:REM  Final de subrutina de entrada del operador

2000 PRINT @(1,50),DATE$,SPC(2),TIME$
2010 TIME=0: DBY(71)=0: REM Pone a cero el temporizador
2020 ONTIME 2, 2000: REM Habilita ONTIME
2030 RETI: REM Final de subrutina de interrupción de ONTIME
```

4

PH0. y PH1. - Entradas - salidas

Función Imprime números hexadecimales de 2 y 4 dígitos

Sintaxis PH0. *expr, expr...*

Vea también HEX$, OCTHEX$

Uso Las declaraciones de PH0. y de PH1. funcionan igual que la declaración PRINT excepto que los valores salen en formato hexadecimal.

La declaración PH1. imprime siempre cuatro dígitos hexadecimales mientras que la declaración PH0. suprime los dos ceros a la izquierda si el número que se imprime es menor de 256 (0100H). El carácter "H" se imprime después del número para identificar el número como hexadecimal.

Los valores impresos por las declaraciones PH0. y PH1. se truncan a un número entero. Si el número que se imprime no está dentro del rango de un número entero válido (0-65535 inclusive), entonces BASIC colocará el formato por defecto de la declaración PRINT.

Ejemplo

```
> 5 FOR I=1 TO  2
> 10 INPUT "NÚMERO HEXADECIMAL ", H
> 20 PRINT1 H
> 30 INPUT "NÚMERO DECIMAL", D
> 40 PH0. D: PH1. D
> 45 PRINT
> 50 NEXT I
> RUN

NÚMERO HEXADECIMAL
?0A5H
165
NÚMERO DECIMAL
?250
FAH
00FAH

NÚMERO HEXADECIMAL
?32H
50
NÚMERO DECIMAL
?257
101H
0101H
```

PICK - Entradas - salidas

Función Opera en números enteros de 16 bits, en un byte o nibble o bit.

Sintaxis PICK (*variable, porción*) = *expresión*
variable = PICK (*expresión, porción*)

Vea también BITS

Uso La instrucción PICK asigna el valor de la *expresión* a la *porción* especificada de una variable numérica. Solamente la *porción* especificada de la *variable* es afectada por PICK. El resto de los bits no cambia. Si el valor de la *expresió*n no cabe en la *porción* especificada de la *variable* entonces aparecerá un mensaje de error BAD ARGUMENT.

El operador PICK devuelve la *porción* especificada de la *expresión* y la asigna a una *variable* numérica. PICK devuelve los valores verdaderos (0FFFFH) y falsos (0) de bits para uso en expresiones de comparación.

porción puede especificar una posición de bit, un nibble (grupo de 4 bits), un BYTE (grupo de 8 bits) o una palabra (16 bits).

Use "B(n)" para especificar una de 16 posiciones de bit, donde n = 0-15.

Use "N(n)" para especificar uno de cuatro nibbles, donde n = 0-3.

Use "H" para escoger el byte más significativo o use "L" para escoger el byte menos significativo .

Use "B" para especificar una conversión de palabra hexadecimal a BCD.

Ejemplo Desmembrar un valor de 16 bits

```
10 Registro = 1120H
20 PH1. "Registro =", REG, "en hexadecimal"
30 PRINT1 "1r bit =", PICK(REG, N(0)), SPC(5),
40 PRINT1 "3r bit =", PICK(REG, N(2))
50 PRINT1 "Valor en binario ="; : FOR BT=0 TO 15
60 IF PICK(REG, B(BT)) THEN 62 ELSE GOTO 64
62 P=NOT(P) : PRINT1 "1"; : GOTO 70
64 PRINT1 GOTO "0"
70 NEXT BT
80 IF P THEN $(0)="impar" ELSE $(0)="par "
90 PRINT1 "La palabra contiene una cantidad ", $(0), "de bits 1"
95 PH1. REG, "o", REG, "tratado como BCD =",
96 PRINT1 PICK(REG, B), "decimal"
100 HB = PICK(REG, H): REM intercambia los bytes
110 PICK(REG, H)=PICK(REG, L): PICK(REG,L)=HB
120 PRINT "Registro con bytes intercambiados =", REG
> RUN
```

Registro = 1120H en hexadecimal
1r bit = 0   3r bit = 1
Valor en binario= 0000010010001000
La palabra contiene una cantidad impar de bits 1
1120H o 4384 tratado como BCD = 1120 decimal
Registro con bytes intercambiados = 2011

POP - Operador avanzado

Función Recupera un valor desde el "stack"

Sintaxis POP *variable*, *variable...*

Vea también PUSH

Uso La declaración POP recupera un valor de la parte superior del "stack" del argumento y lo asigna a una *variable*. La última *variable* en la lista de variables de la declaración POP tendrá asignada el último valor desde el "stack" del argumento.

Ejemplo Ver el ejemplo de PUSH

PRINT - Entradas - salidas

Función Transmite datos saliendo del puerto serial especificado

Sintaxis PRINT*n, expr. expr ...*

Abreviatura P., P1., P2., P3.

Vea también
- TAB Colocación de cursor absoluta en la línea corriente
- SPC Colocación de cursor relativa en la línea corriente
- CR Devuelve el cursor a la posición 1 en la línea corriente (ningún LF)
- @(y, x) Colocación de cursor absoluta en la pantalla ANSI
- USING Para alinear las comas de números impresos con PRINT y para imprimir un número especificado de caracteres de una variable de string.

Uso PRINT transmite datos saliendo del puerto serial especificado por *n*.

expr puede ser una string o expresión o constante numérica. Se pueden hacer salir varios valores en una sola declaración PRINT si son separados por comas.

Normalmente se envía un carácter de **retorno de carro** y de **avance de línea** en el fin de cada PRINT. Esta función puede ser suprimida agregando una coma en el final de la declaración PRINT.

Las declaraciones PRINT del programa se pueden comenzar y parar desde dispositivos externos con XOFF < *CTRL-S* > y XON < *CTRL-Q* > o con la entrada CTS del "handshake" de hardware. Vea la página 4.84 de SETPORT para más información sobre Control de flujo del puerto serial.

Ejemplo

```
> 10 FOR I=1 TO  3: PRINT1 I,: NEXT I: PRINT1 -5
> RUN

123-5

READY
> PRINT1 2**16-1, "BYTES DE MEMORIA! (", 65.535E3,")"
65535 BYTES de MEMORIA! (65535)
READY
>
```

Uso especial Use el operador CHR$ para imprimir los códigos especiales de 8 bits que no son una parte del conjunto de caracteres estándares ASCII mostrado en el apéndice D, pero esto depende del aparato receptor. Por ejemplo, PRINT CHR$(219) imprimirá un signo dollar $ en una PC IBM.

Use PRINT USING(\n \), $(var) para imprimir los primeros *n* caracteres de una variable string cuando los caracteres delimitadores de string ASCII, **null** (ASCII=0) y **retorno de carro** (ASCII=13) están contenidos dentro del string como valores de datos (o use CHR$ para imprimirlos explícitamente).

PUSH - Operador avanzado

Función Coloca un valor en el "stack"

Sintaxis PUSH *expresión*

Vea también POP

Uso PUSH (EMPUJA) coloca el valor de la *expresión* en la parte superior del "stack" del argumento. El valor de la última *expresión* en la lista de expresiones de la declaración PUSH será el último valor puesto en el "stack".

El PUSH y el POP son convenientes para pasar valores a y desde subrutinas de propósito general.

Ejemplo Este ejemplo usa PUSH y POP para pasar datos a una subrutina de propósito general que realice la tarea repetitiva de separar cuatro dígitos BCD.

```
> 10 PUSH 700 + 53
> 20 GOSUB 100
> 30 TRANSFER(128): TRANSFER(LSB)
> 31 REM Envía 4 dígitos BCD al PLC
> 40 TRANSFER(129): TRANSFER(MSB)
> 50 INPUT "VALOR PARA EL PAR  DE REGISTROS 400/401?”, $(0)
> 60 PUSH VAL($(0))
> 70 GOSUB 100
> 80 PRINT1 "REGISTRO 400 =", LSB
> 85 PRINT1 "REGISTRO 401 =", MSB
> 90 END
> 100 POP  D: MSB = INT(D/100)
> 101 REM los dos dígitos más significativos
> 110 LSB = D-INT(D/100)
> 111 REM los dos dígitos menos significativos
> 120 RETURN
> RUN

VALOR PARA EL PAR  DE REGISTROS 400/401? 9642
Registro 400 = 42
Registro 401 = 96
```

TRANSFER es una instrucción usada en solamente algunos módulos...

READ - Entradas - salidas

Función Asigna valores constantes de la declaración DATA a variables

Sintaxis READ *variable, variable...*

Vea también DATA, RESTORE

Uso READ asigna el valor de una expresión numérica especificada en una declaración DATA a la *variable*.

Varias *variables* en la lista de READ son separadas por comas.

A la primera *variable* en la primera declaración READ en el programa se le asigna el valor de la primera expresión en la primera declaración DATA en el programa. Cada *variable* adicional encontrada en una declaración READ se le asigna el valor de la expresión siguiente en una declaración DATA.

Las declaraciones DATA aparecen a las declaraciones READ como una lista larga de expresiones.

Si se ha leído la última expresión en la declaración DATA y se ejecuta otra declaración READ, BASIC parará la ejecución del programa con el error, NO DATA - In LINE xx.

4

Ejemplo

```
> STRING 8001, 79: REM asigna espacio para 100 * 79 strings
> 10 REM Carga o copia códigos de error
> 20 FOR CODIGO = 1 TO  4
> 30 READ ERR(CODIGO)
> 40 NEXT CODIGO
> 50 DATOS 2, 4, 7, 22
> 60 $(2) = "Sin papel"
> 70 $(4) = "El alimentador está con nivel bajo"
> 80 $(7) = "Presión demasiada baja"
> 90 $(22) = "No puede seguir sin el pago final de"
> 91 $(22)=$(22)+"esta máquina "
```

REM - Misceláneo

Función Permite colocar comentarios no ejecutables

Sintaxis REM *Comentarios*

Uso La declaración REM se usa para agregar *comentarios* a un programa. Todo lo que está en una línea después de la instrucción REM se ignora en BASIC.

El hecho de que la declaración REM es ejecutable en el modo COMMAND puede ser útil en ciertas aplicaciones. Si se usa una computadora para cargar programas en el módulo ASCII/BASIC, las declaraciones REM sin el número de líneas se podrían incluir en la versión del programa de la computadora, con todo no aparecerían en el programa del módulo BASIC. Esto permitiría que el programa principal fuera documentado sin consumir memoria en el sistema del módulo.

ABM Commander Plus lleva este concepto un paso más adelante y es que opcionalmente carga programas sin todos los *comentarios* en el programa. Las líneas de programa que comienzan con REM *comentarios* se reducen apenas al número de línea y REM de modo que el número de línea se puede usar en declaraciones GOTO y GOSUB.

Ejemplo

```
> 10 REM hace salir el código
> 20 PRINT1 CD
> 30 IF INLEN2 = 0 THEN GOTO 30 : REM  Espera por una entrada
```

RESTORE - Entradas - salidas

Función Permite que los valores constantes de la declaración DATA se puedan leer con la declaración READ otra vez

Sintaxis RESTORE

Vea también DATA, READ

Uso Coloca el puntero usado por READ al inicio de los datos DATA. Después de RESTORE, la siguiente variable de READ será asignada al valor de la primera expresión en la primera declaración de DATA en el programa.

Ejemplo

```
10 Rem Usando DATA-READ-RESTORE
11 REM para definir una pseudo función
20 REM el argumento de la función se pasa a la función en WRD
30 WRD = 4598
40 RESTORE: READ MSB, LSB
50 PRINT "El par de registros 413/412 es",MSB, "/",LSB
60 WRD = 248
70RESTORE: READ ALTO, BAJO
80 PRINT "Los dos dígitos más significativos BCD son", ALTO
90 PRINT  "Los dos dígitos menos significativos BCD son", BAJO
120 DATA  INT(WRD/100), WRD - INT(WRD/100)*100
> RUN

El par de registros 413/412 es 45/98
Los dos dígitos más significativos BCD son 2
Los dos dígitos menos significativos BCD son 48
```

RETI - Interrupción

Función Marcar el final de una subrutina de manejo de interrupción

Sintaxis RETI

Vea también IDLE, ONPORT, ONTIME

Uso RETI se usa para salir de las subrutinas de interrupción especificadas por las declaraciones ONTIME o ONPORT.

RETI realiza una función similar a la declaración RETURN y además identifica el final de la subrutina de interrupción para poder reconocer otra vez interrupciones.

Si el usuario no ha ejecutado la declaración RETI en la subrutina de interrupción, serán ignoradas todas las interrupciones futuras (vea también CLEAR I).

RETURN - Control de flujo

Función Marcar el final de una subrutina

Sintaxis RETURN

Vea también GOSUB

Uso RETURN se usa para marcar el final de una subrutina y causa que el flujo del programa se reanude con la declaración que sigue a la declaración GOSUB ejecutada recientemente. La secuencia GOSUB-RETURN puede ser usada unas dentro de otra (nested). Es decir, las subrutinas pueden llamar otras subrutinas conforme a la limitación del tamaño del "stack" de control.

4

Ejemplo

```
> 10 FOR I=1 TO 5
> 20 GOSUB 50
> 30 NEXT I
> 40 END
> 50 PRINT1 I, SPC(1),
> 60 RETURN
> RUN

1 2 3 4 5

> 10 FOR I=1 TO  5: GOSUB 50
> 20 I NEXT: END
> 30 A=I*I
> 40 RETURN
> 50 GOSUB 30: PRINT1 I, SPC(1), A, SPC(1),
> 60 RETURN
> RUN

1 1 2 4 3 9 4 16 5 25
```

REVERSE$ - Operador de string

Función REVERSE$ vuelve una cadena de n caracteres comenzando con el último carácter.

Sintaxis *variable de string* = REVERSE$(*expresión de string*, *n*)

Vea también LEFT$, MID$, RIGHT$

Uso *n* es una expresión y especifica el número de caracteres de la *expresión de string* que se asignará a la *variable de string*.

n debe estar en el rango 0 a 254. REVERSE$ devuelve una string o cadena de caracteres que consiste en el último carácter hasta el enésimo carácter de la *expresión de string*. Si *n* es mayor o el igual a la longitud de la *expresión de string* entonces se devuelve toda la *expresión de string* . Si *n* es 0 entonces REVERSE$ devuelve el string sin caracteres (Null).

REVERSE$ permite que usted invierta la orden de todo o una parte de una string en una sola declaración.

Ejemplo

```
> PRINT1 REVERSE$("SÉVER LA",20)
AL REVÉS

> PRINT1 REVERSE$("N20G45", 2)
54
```

RIGHT$ - Operador de string

Función RIGHT$ devuelve una string comenzando con el enésimo carácter desde el último carácter

Sintaxis *variable de string* = RIGHT$(*expresión de string*, *n*)

Vea también LEFT$, MID$, REVERSE$

Uso *n* es una expresión y especifica el número de caracteres de la *expresión de string* que se asignará a la *variable de string*.

n debe estar en el rango 0 a 254. RIGHT$ devuelve una string que consiste en el enésimo hasta el último carácter de la *expresión de string*. Si *n* es mayor o el igual a la longitud de la *expresión de string* entonces se devuelve toda la *expresión de string*. Si *n* es 0 entonces RIGHT$ devuelve el carácter **null**.

RIGHT$ permite que usted escoja el final de un string.

Ejemplo

```
> PRINT1 RIGHT$("BASIC AMPLIADO DE FACTS", 5)
FACTS
```

Usando MID$ y LEN para alcanzar el mismo resultado que RIGHT$

```
> $(0)="FIN DE SEGMENTO "
> PRINT1 MID$($(0), LEN($(0))-6)
 SEGMENTO  (note que hay un espacio antes de la S de SEGMENTO)
```

4

RND - Operador matemático

Función Calcular un número pseudo aleatorio en el rango entre 0 y 1, incluso 0 y 1.

Sintaxis RND

Uso Devuelve un número pseudo-aleatorio en el rango entre 0 y 1 inclusive. El operador RND utiliza una semilla binaria de 16 bits y genera 65536 números pseudo-aleatorios antes de repetir la secuencia. Los números generados están específicamente entre 0/65535 y 65535/65535 inclusive.

A diferencia de la mayoría de los programas BASIC, el operador RND en este BASIC no requiere un argumento o un argumento simulado. De hecho, si se pone un argumento después del operador RND, se genera un error de SINTAXIS.

Ejemplo

```
PRINT RND
.30278477
```

SETINPUT - Entradas - salidas

Función Configurar la declaración INPUT

Sintaxis SETINPUT *no hay eco, no edit, terminador, longitud, espera primer, espera último*

Vea también INPUT, SETPORT

Uso SETINPUT establece los parámetros operacionales para las declaraciones subsecuentes INPUT. Cuando está INPUT sin argumentos, será generado un mensaje que recuerda al usuario la sintaxis de SETINPUT (En inglés).

no hay eco, el único parámetro de SETINPUT que no es opcional y debe ser un 0 o un 1. Si *no hay eco* es 1 entonces los caracteres recibidos por la declaración INPUT no serán repetidos. Cuando *no hay eco* es 0, INPUT repetirá todos los caracteres recibidos. El valor por defecto es 0, eco.

no edit es un solo carácter, un 0 o un 1. Si *no edit* es 1 entonces los caracteres backspace (ASCII 8), *CONTROL-D* (ASCII 4), y Delete (ASCII 127) estarán deshabilitados y todos los caracteres de control (ASCII 0 a 31) serán INPUTS (se ignoran los caracteres de XON/XOFF cuando el "Handshake"del software está activado). Si *no edit* es 0 se permite modificar INPUTS y el resto de los caracteres de control serán ignorados. Esto permite borrar un carácter de entrada anterior con INPUT. El valor por defecto es 0, permite BS/DEL.

el *terminador* es cualquier carácter ASCII, 0 a 255. INPUT para cuando se recibe el carácter *terminador*. Si el *terminador* es 0 entonces la verificación del carácter final de la entrada es desactivado. El adaptador por defecto es un **retorno de carro** (ASCII 13).

la *longitud* es una expresión que especifica la cantidad máxima de caracteres (por string) que recibirá INPUT. La entrada INPUT se detiene si la cantidad de caracteres recibidos es igual a la *longitud*. El rango de la *longitud* es 0 a 255. Si la *longitud* es 0 o 255 entonces se aceptan 255 caracteres y un carácter BELL (ASCII 7) será repetido si se transmiten más de 255 caracteres. El valor por defecto de la *longitud* es 0.

espera primer es una expresión de número entero, 0 a 65535, que especifica el tiempo máximo en milisegundos que la declaración INPUT esperará el recibo del primer carácter. Si un carácter no se recibe dentro del tiempo especificado entonces BASIC reanudará la ejecución con la declaración que sigue la declaración INPUT. Si *espera primer* es 0 entonces la declaración INPUT esperará indefinidamente un carácter. Éste es el valor por defecto.

espera último es una expresión de número entero, 0 a 65535, que especifica el tiempo máximo en milisegundos que la declaración INPUT esperará el recibo de cada carácter subsecuente a haber recibido el primero. Si no se recibe otro carácter dentro del tiempo especificado entonces BASIC reanudará la ejecución con la declaración que sigue a la declaración INPUT. Si *espera último* es 0 entonces la declaración INPUT no tendrá un time-out. Éste es el valor por defecto.

Ejemplo 1

```
10 REM INPUT no tiene eco de los caracteres
20 SETINPUT 1
```

Ejemplo 2

```
10 REM siempre entre con INPUT 3 caracteres
20 SETINPUT 0, 0, 0, 3
```

Ejemplo 3

```
10 TERM = 61:REM configura el caracter de terminación de INPUT
11 REM como "="
20 ESPERA1 = 3000: REM  Time-out si no hay INPUT en 3 segundos
30 ESPERA2 = 100: REM Time-out si no más INPUT en 1 segundo
40 SETINPUT 1,0, TERM, 79, ESPERA1, ESPERA2
```

Ejemplo 4

```
05 REM ENTRE un carácter sin repetir en el plazo de 60 segundos
10 SETINPUT 1, 0, 0, 1, 60000
20 INPUT2  "Apriete cualquier tecla para continuar...",$(0)
```

SETPORT - Entradas - salidas

Función Configurar un puerto de comunicaciones.

Sintaxis SETPORT *puerto, baud, paridad, bits de datos, bits de stop, handshake, multidrop*

Vea también INPUT, SETINPUT

Uso SETPORT especifica la velocidad en Baud, la estructura y el Control de flujo para un puerto serial. Cuando SETPORT se entra sin argumentos se genera un mensaje que le recuerda al usuario la sintaxis y las opciones de SETPORT.

Puesto que cada puerto serial tiene un buffer independiente de 255 carácteres, se pueden recibir los datos desde los dispositivos seriales externos al mismo tiempo que el módulo de BASIC está realizando otra tarea tal como un cálculo de lazo de PID o está entrando valores de memorias de la CPU. En algunos usos intensivos de comunicación el número de caracteres en cada buffer de entrada se debe examinar por el programa principal periódicamente para poder entrar datos con INPUT antes de que se llene un buffer (vea la declaración INLEN).

puerto indica qué puerto serial se está configurando. El *puerto* es el único argumento de SETPORT que no es opcional y debe ser 1, 2, o 3 dependiendo de qué módulo tiene usted . Cada uno de los puertos puede ser configurado diferentemente y conserva su configuración hasta que se ejecuta otra declaración de SETPORT. Si no se usa SETPORT entonces los puertos seriales toman los valores siguientes: ninguna paridad, 7 bits de datos, 1 bit de STOP, y ningún handshake. La velocidad por defecto es establecida por AUTOSTART.

baud es una expresión que especifica la tasa de transacciones de comunicación. SETPORT no verifica que la velocidad especificada sea "válida". Las velocidades típicas son 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115.200. Vea su manual de usuario específico del módulo para determinar a qué velocidades puede funcionar su módulo.

la *paridad* es un solo carácter o A (*dirección remota, máscara*) que especifica el estado del bit de paridad según lo mostrado abajo.

O paridad impar

E Paridad par

N Ningún bit de paridad (si el bit de paridad es 7 bits de datos entonces se ignoran los caracteres recibidos y 0 en caracteres transmitidos)

A permite la función automática del reconocimiento de dirección (AAR o Automatic Address Recognition). La *dirección remota* es una expresión en el rango de 0 a 255 que especifica la dirección de estación remota o esclava del módulo.

la *máscara* (Mask) es opcional. Si la máscara no se especifica, entonces el módulo recibirá solamente datos después del recibo del byte remoto o del byte 255 de la dirección de Broadcast.

Los ceros en el byte de la máscara definen posiciones de bit "que no importan"en el byte remoto de la dirección para permitir al módulo recibir los datos para un grupo de direcciones.

Al habilitar la función AAR se fuerza una estructura de datos de 11 bits con una palabra de 8 bits y 1 bit de STOP. El noveno bit de paridad se usa para distinguir entre la dirección y los bytes de datos. El hardware en el módulo de ASCII/BASIC comprueba el noveno bit de paridad. Si es un 1 entonces el byte de la dirección recibido se compara a la dirección remota (modificada opcionalmente con la máscara). Observe que esta comparación de la dirección ocurre en hardware, no en software. Si es igual entonces el byte de dirección y los bytes de datos subsecuentes se cargan en buffer de entrada.

los *bits de datos* especifican la cantidad de bits de datos y deben ser 7 u 8. La paridad recibida se ignora en el puerto 2 y el puerto 3 para palabras de 8 bits.

los *bits de stop* especifican el número de bits de stop y deben ser 1 o 2. Observe que 7 bits de datos y 2 bits de stop son iguales que 7 bits de datos marcan paridad y 1 bits de stop. Asimismo, 8 bits de datos y 2 bits de stop son iguales que 8 bits de datos marcan paridad y 1 bit de stop.

el "*Handshake*" es un solo carácter que especifica el Control de flujo de la comunicación según lo mostrado abajo.

S o T "Handshake"de software (XON/XOFF en base a carácter por carácter)

H "Handshake" bidireccional de hardware CTS/RTS

U Control de flujo unidireccional de hardware con CTS

N Ningún "Handshake"

Si ningunas de las opciones antedichas de "Handshake"son apropiadas para su aplicación entonces considere el Control de flujo BASIC usando a los operadores DTR y DSR. La salida RTS RS-232 se controla con el operador BASIC DTR. El estado de la entrada CTS RS-232 es mostrado por el operador BASIC DSR (Vea DSR y DTR).

Especifique una "M" para *multidrop* para permitir que funcionen los transmisores RS-422/485 solamente cunado se hace un PRINT.

Especifique una "P" para *peer to peer* para indicar dejar los transmisores RS-422/485 ON incluso cuando no se active PRINT. Vea el manual de usuario para los ejemplos de cableado.

"Handshake" por Software

El carácter de Control de flujo de software < *CTRL-S* > (XOFF, ASCII 19) es usado por un dispositivo externo para decir al módulo parar la impresión (PRINT). Cuando el buffer de recepción en el dispositivo externo se descarga suficientemente, transmite una señal <*CTRL-Q*> (XON, ASCII 17) al módulo, diciéndole que continúe transmitiendo. El "Handshake"por software funciona con cada carácter (no en línea por línea como BASIC de MCS-51 de Intel). Es decir, BASIC ampliado de FACTS Eng. verifica para ver si XOFF fue recibido antes de imprimir (PRINT) el próximo carácter.

El "Handshake"por software se usa a menudo con terminales (ABM Commander Plus lo usa), las impresoras y módems externos cuando las transmisiones seriales son todas 7 bits ASCII.

"Handshake"Bidireccional de Hardware CTS/RTS

El "Handshake"bidireccional de hardware CTS/RTS funciona carácter por carácter y se usa típicamente con módems externos o cuando hay datos de 8 bits en transmisiones seriales. En este caso, no puede ser usado el Control de flujo de software porque los caracteres de Control de flujo de software XON (ASCII 17) y XOFF (ASCII 19) pueden aparecer como datos en la s transacciones de comunicación.

La señal RTS es una salida del módulo de BASIC que se hace activa al comienzo de una declaración PRINT. Está solicitando la confirmación del dispositivo externo para hacer una transmisión (digamos que ABM está listo para hacer un PRINT). La señal CTS es una entrada al módulo de BASIC que cuando está ON indica que el módulo puede comenzar a transmitir al dispositivo externo (le dice, muy bien, puede hacer PRINT).

En el final de la declaración PRINT la señal RTS se desactiva después de aproximadamente dos tiempod de un carácter para velocidades menores de 9600 Baud. Si la velocidad es 9600 o más alta entonces la señal RTS se desactiva inmediatamente después que se transmite el último carácter en la declaración PRINT.

Si el módulo BASIC no ve la señal CTS en el plazo de 1 segundo de hacer la señal RTS ON entonces la señal RTS se hace OFF y el modo de "Handshake"se cambia a ninguno (N).

Control de flujo unidireccional de Hardware de CTS

El Control de flujo unidireccional de hardware CTS funciona carácter por carácter y se usa típicamente con módems externos.

La señal CTS es una entrada al módulo de BASIC que el módem hace OFF para mandar al módulo parar el envío de datos. El módem hace ON la señal CTS para instruir al módulo BASIC de reanudar el envío de datos. El Control de flujo unidireccional de hardware CTS es exactamente como el Control de flujo bidireccional de hardware RTS/CTS excepto que la señal RTS del módulo está siempre ON.

Ningún "Handshake"

Al interconectar dispositivos que no apoyan ningún tipo de handshake, el usuario debe tener cuidado que los caracteres no se pierdan en una transmisión. Las impresoras, por ejemplo, se pueden interconectar fácilmente al módulo BASIC seleccionando una velocidad suficientemente lenta para dar un tiempo adecuado para que la impresora continúe.

Ejemplo

```
05 REM Configure el puerto 1 para comunicación con un terminal
10 SETPORT 1, 9600, N, 7, 1, S
14 REM configura el puerto 2 para comunicación con
16 REM un transmisor binario
20 SETPORT 2, 1200, O, 8, 1, N
```

Ejemplo

Configure el puerto 1 con 9600 Baud, ninguna paridad, la palabra de 8 bits, 1 bit de STOP, "Handshake"por software XON/XOFF, y el modo de multi-nodos RS-422/485.

```
SETPORT 1, 9600, N, 8, 1, S, M
```

Ejemplo

El ejemplo siguiente muestra la característica AAR (reconocimiento automático de la dirección) incorporado de hardware.

Estación esclava

```
1000 string2551,254: Rem 10, 254 strings
1010 RA=2: REM Activa AAR y define nuestra dirección remota en 2
1030 SETPORT 1,9600,A(RA),8,1,N,M
1040 REM interrumpe programa principal para entrar (INPUT) los datos para RA
1050 INLEN1=0: ONPORT 1,1070
1060 GOTO 1240 : REM ejecuta el programa principal
1070 REM Apaga el eco, ENTRA (INPUT) todo ASCII, sin carácter de términación
1080 datos REM tamaño de bloque máximo de  = 254 bytes
1090 REM espera para siempre por el primer carácter
1100 REM Entra datos hasta que el espacio entre caracteres > 1 vez el tiempo del
carácter
1110 SETINPUT 1,1,0,0,0,(11/9.600)
1120 REM entra el bloque de datos de la estación maestra
1130 INPUT1, $(0)
1140 GOSUB 1230: REM procesa los datos de la estación maestra
1150 REM si  la dirección de broadcast no responde
1160 IF ASC($(0), 1)=255 THEN GOTO 1190
1170 REM  envía respuesta a la estación maestra aquí
1180 PRINT1 CHR$(1), CHR$(6), CHR$(RA),: REM UN ACK
1190 REM interrupción al programa principal de los datos de ENTRADA para RA
1200 ONPORT 1,1070
1220 RETI: REM vuelve al programa principal
1230 RETURN: REM Nada a procesar en este ejemplo
1240 REM Comienza el lazo principal del programa
1250 DO
1260 REM Nada
1270 UNTIL 1=0
```

Estación Principal

```
1000 RA=1: REM Activa AAR y establece nuestra dirección remota
1020 SETPORT 1,9600,A(RA),8,1,N,M: SETPORT 2,9600,N,8,1
1030 REM con AAR permitido, el primer byte de una  declaración PRINT tendrá
1040 REM  el noveno bit ON. Éste es la dirección de la estación esclavo
1050 REM. El bloque de los datos puede ser imprimibles
1052 REM ASCII o ASCII hexadecimal
1060 PRINT1 CHR$(2), "Mensaje para la estación 2 solamente",
1070 STA=2: GOSUB 1120: REM Obtiene un ACK(reconocimineto) del esclavo 2
1080 PRINT1 CHR$(3), "Mensaje para la estación 3 solamente",
1090 STA=3: GOSUB 1120: REM consigue un ACK del esclavo 3
1100 PRINT1 CHR$(255), "Mensaje para todas las estaciones",
1110 GOTO 1030: REM Lazo
1120 REM espera por la respuesta de esclavos
1130 SETINPUT 1,1,0,254,200,(11/9.600)
1140 INPUT1, $(0)
1150 IF INPLEN=0 THEN PRINT2 "ninguna respuesta de estación", STA : RETURN
1160 IF ASC($(0), 2)=6 THEN GOTO 1162 ELSE GOTO 1170
1170 1162 PRINT2 "ACK DEL ESCLAVO",  ASC($(0), 3)
RETURN
```

Avanzado Usando máscara para definir grupos de direcciones de estación esclava

El reconocimiento automático de dirección (AAR) reduce el tiempo de la CPU requerido para mantener comunicaciones seriales. Puesto que la CPU se interrumpe solamente cuando recibe su propia dirección, se eliminan tiempos adicionales de software para comparar direcciones.

Una vez que AAR sea habilitado por la declaración SETPORT, se configura el noveno bit del primer byte de cada declaración PRINT. Este byte es la dirección del esclavo blanco, a ser usado en las transacciones de comunicación. El noveno bit es borrado por los caracteres restantes en la declaración PRINT.

Cuando un esclavo blanco recibe un byte de dirección correcto o igual, ese byte y todos los bytes subsecuentes se cargan en el buffer de entrada. Al completarse una declaración INPUT, el esclavo vuelve automáticamente al modo de hardware AAR. La ejecución de una declaración INLEN1=0 también borrará el buffer de entrada y bloquea la recepción posterior de datos hasta que se recibe otro byte de dirección igual.

El maestro puede comunicarse con todos los esclavos usando la dirección de broadcast (255). El maestro puede comunicarse selectivamente con los grupos de esclavos usando una dirección de grupo.

La dirección individual de esclavos es especificada por la *dirección remota*. El byte opcional de *máscara* define posiciones de bit “*que no importan*” en la dirección remota, lo que proporciona así la flexibilidad de direccionar uno o más esclavos a la vez.

Si el número de ceros en el byte de la máscara es igual a N, entonces el número máximo de esclavos en el grupo es 2**N.

Ejemplo

Estación Auxiliar 1:

dirección remota	**1111 0001 (0F1H)**
máscara	**1111 1010 (0FAH)**
Dirección de Grupo	**1111 0x0x(0F0H, 0F1H, 0F4H, 0F5H)**

Estación Auxiliar 2:

dirección remota	**1111 0011 (0F3H)**
máscara	**1111 1001 (0F9H)**
Dirección de Grupo	**1111 0xx1 (0F1H, 0F3H, 0F5H, 0F7H)**

Estación Auxiliar 3:

dirección remota	**1111 0000 (0F0H)**
máscara	**1111 1100 (0FCH)**
Dirección de Grupo	**1111 00xx (0F0H, 0F1H, 0F2H, 0F3H)**

La dirección única del esclavo 1 es 0F4H. La dirección única del esclavo 2 es 0F7H. La dirección única de la estación auxiliar 3 es 0F2H.

La *dirección del grupo* para los 3 esclavos es 0F1H. La *dirección del grupo* para las estaciones 1 y 2 es 0F5H. La *dirección del grupo* para las estaciones 2 y 3 es 0F3H y la *dirección del grupo* para las estaciones 1 y 3 es 0F0H.

SGN - Operador matemático

Función Devuelve el signo de la expresión

Sintaxis SGN(*Expresión*)

Uso Devolverá un valor de +1 si la *expresión* es mayor que cero, cero si la *expresión* es igual a cero, y -1 si la *expresión* es menor de cero.

Ejemplo

```
PRINT SGN(52)
1

PRINT SGN(0)
0

PRINT SGN(-8)
-1
```

SIN - Operador matemático

Función Calcula el seno de la expresión

Sintaxis SIN(*expresión*)

Uso Calcula el seno de la *expresión*, que está en radianes. Los cálculos se realizan a 7 dígitos significativos. La *expresión* debe estar entre +20000 y -200000.

Ejemplo

```
PRINT SIN(0)
0

PRINT  SIN(60*3.14/180)
.8657599
```

SPC - Entradas - salidas

Función Utilizado en la declaración PRINT para hacer salir una cantidad de espacios

Sintaxis PRINT SPC (*cantidad*)

Vea también PRINT, CR, TAB, USING, @(*línea*, *columna*)

Uso *Cantidad* es una expresión que especifica cuantos caracteres de espacio serán usados con PRINT (0-255).

El SPC es usado para poner espacios adicionales entre salidas de valores por la declaración PRINT.

Ejemplo

```
10 $(0)="TEMPERATURA"
15 $(1)="PRESIÓN"
20 PRINT1 $(0), SPC(4), $(1)
30 T_POS=LEN($(0))/2-3
40 P_POS=LEN($(0))+4+LEN($(1))/2-3
50 PRINT1 USING(###.##), SPC(T_POS), A, SPC(P_POS), B
TEMPERATURA              PRESIÓN
123.45                   345.67
```

SQR - Operador matemático

Función Calcula la raíz cuadrada de la expresión

Sintaxis SQR(*expresión*)

Uso Calcula la raíz cuadrada de la *expresión*. *expresión* no puede ser menor de cero. El resultado devuelto será exacto dentro de una tolerancia de + o - un valor de 5 en el menor dígito significativo.

Ejemplo

PRINT SQR(9)	**PRINT SQR(45)**	**PRINT SQR(100)**
3	**6.7082035**	**10**

STOP - Control de flujo

Función Suspende la ejecución del programa

Sintaxis STOP

Vea también END, CONT

Uso STOP se usa para detener la ejecución del programa. Después de que se haya parado la ejecución del programa, las variables pueden ser mostradas y ser modificadas. La ejecución del programa puede ser reanudada donde ha parado con el comando CONT. La declaración STOP permite el programar “búsqueda de errores (debugging)” en forma fácil.

Ejemplo

```
> 10 FOR I=0 TO 9: D(I)=I: NEXT
> 15 STOP
> 20 FOR I=0 TO 9: PRINT1 D(I), SPC(1),: NEXT
> RUN

STOP    - In line 20
READY
> FOR I=5 TO 9: P. D(I),: NEXT
56789
> D(9)=0

> CONT

0 1 2 3 4 5 6 7 8 0

READY
>
```

STORE@ o ST@ - operador avanzado

Función Almacena un número de coma flotante de seis bytes en la dirección especificada.

Sintaxis STORE@ *dirección*

Vea también BYTE, WORD, LOAD @

Uso La declaración STORE@ permite que el usuario almacene números de coma flotante en cul;aquier lugar en la memoria de datos.

La *dirección* es la posición de memoria más alta donde el número va a ser almacenado. El número que se almacenará se debe poner primero en el "stack" del argumento con la declaración PUSH.

Ejemplo Vea el CAPÍTULO 8, AVANZADO

STR$ - Operador de string

Función STR$ vuelve el equivalente de string de una expresión matemática

Sintaxis *variable de string* = STR$(expresión matemática)

Vea también VAL

Uso STR$ convierte la *expresión matemática* en una string decimal equivalente que se asigne a la variable de string.

Ejemplo

```
P. STR$(+123.4)
123.4

P. STR$(-.002)
-.002

P. STR$(3.1415926*10**-6)
.0000031415926

P. STR$(80H)
128

P. STR$(-12 E10)
-120000000000
```

4

STRING - Gerencia de memoria

Función Asignar la memoria para almacenaje de string

Sintaxis **STRING** (*total, longitud*)

Uso La declaración STRING asigna la memoria para las variables alfanuméricas de string casi de la misma manera que la declaración DIM asigna la memoria para las variables numéricas de arreglos. La declaración STRING especifica la cantidad total de bytes de memoria de datos que serán asignados para el almacenaje de string y la *longitud* máxima de cada string. La *longitud* debe estar en el rango 2 a 254. Cada STRING requiere un byte de memoria por cada carácter en la STRING y además un byte adicional.

La fórmula siguiente se puede usar para determinar la memoria total necesitada para almacenar una *cantidad* dada de strings con una longitud de string máxima particular.

*total = (longitud + 1) * cantidad + 1*

En vista de la cantidad de memoria de datos disponible por el usuario para el almacenaje variable, una forma de asignar la memoria para almacenaje de strings es “asegurarse de que es bastante".

Si la declaración STRING se usa en el programa debe estar después de la declaración MTOP pero antes de DIM. Esto es, porque STRING primero borra la memoria hasta MTOP antes de asignar el espacio de almacenaje del string. La única manera de desasignar la memoria para el almacenaje de string es con una declaración de string 0,0 (NEW, CLEAR y RUN borra las variables de string pero no libera la memoria asignada por STRING).

Ejemplo Asignar la memoria para 100 strings con hasta 79 caracteres en cada secuencia.

```
> P. (79+1)*100+1
8001

> 10 STRING 8001.79
```

Especial Si van a ser conservadas las variables y los datos del programa durante una pérdida de energía (AUTOSTART mode=2, RUN sin CLEAR), entonces se debe entrar la declaración de string una vez como comando antes de hacer funcionar el programa.

```
REM asigna explícitamente memoria para 99, 44 strings
> P. (44+1)*99+1
4456

READY
> STRING 4456, 44
```

SYSTEM - Misceláneo

Función Leer y configurar información del sistema

Sintaxis SYSTEM(*código*) = *expr.*
var = SYSTEM(*código*)

Uso Se usa el comando SYSTEM para tener acceso a varios datos del sistema que se pueden alcanzar usando una declaración o comando BASIC. Los códigos de sistema abajo se apoyan en todos los módulos. Vea al manual de usuario específico del módulo para códigos de sistema adicionales únicos a ese módulo.

Código DESCRIPCIÓN

Código	DESCRIPCIÓN
0	Dirección del comienzo del PRM 0 (primer programa, donde se modifica)
1	Re-imprime el último mensaje de error en modo de comando. Devuelve el número de línea del último error en modo RUN.
2	Si es verdad, entonces agrega caracteres de verificación de error CRC-16 a las declaraciones PRINT y verifica CRC-16 en INPUTS de strings. COMERRn es verdadero si la verificación de error de INPUT falla.
3	Si es verdad, sale el bit menos significativo de CRC-16 primero. Si es falso, sale el bit mas significativo de CRC-16 MSB primero.
4	Valor de milisegundos del temporizador (era DBY(71)).
5	Valor de ERRCHK.
6	Dirección de la primera posición de memoria libre en el banco 1. Ésta es la localización del primer byte después del final de los programas archivados.
7	Selecciona el puerto 1 como el puerto de programación si *expr*. evalúa como 0. Si *expr*. es 1 entonces el puerto 2 se convierte en el puerto de programación. Éste es el equivalente a COMMAND@ durante la ejecución del programa.
8	Devuelve el número de línea siguiente en el programa BASIC. Si SYSTEM(8) está en la última línea del programa devuelve 0. (Vea la declaración GO-PROGRAM).

4

TAB - Entradas - salidas

Función Utilizado en la declaración PRINT para especificar la posición

Sintaxis PRINT TAB (*expresión*)

Vea también PRINT, CR, SPC, USING, @(*Línea, columna*)

Uso La declaración TAB especifica en qué posición comenzar a imprimir el siguiente artículo en la lista de la declaración PRINT. El valor de la *expresión* debe ser menor de 256. Si el cursor (posición de PRINT corriente) está más allá de la posición de TAB especificada, la TAB es ignorada y PRINT comienza en el cursor.

Ejemplo

```
> 10 PRINT TAB(4), "TEMPERATURA",TAB(19), "PRESIÓN"
> RUN
   TEMPERATURA     PRESIÓN

READY
> 10 STRING 8001,79
> 20 FOR I=1 TO  3
> 30 INPUT $(I)
> 40 NEXT
> 50 FOR I=1 TO  3
> 60 PRINT1 TAB(I*3), I*3, $(I),
> 70 NEXT
> RUN

?A
?CDE
?GHIJK
            3A        6CDE9GHIJK tercer TAB ignorado
           "^^"       "^" denota la posición de TAB (no impresa)
```

Especial La posición actual del cursor para propósitos de TAB es determinada contando la cantidad de caracteres impresos desde el último **retorno de carro**. Al enviar secuencias de escape a un terminal de operador, pueden ser evitados problemas usando SPC en vez de TAB para posición de salidas.

La declaración TAB se puede usar con la función de cursor de la pantalla ANSI que es @(*linea, columna*).

La posición actual del cursor no es específica al puerto. Para prevenir problemas de TAB, imprima con PRINT un **retorno de carro** para resetear la posición actual del cursor a cero antes de comenzar una salida con PRINT TAB en un puerto diferente.

TAN - Operador matemático

Función Calcular la tangente de la expresión

Sintaxis TAN(*expresión*)

Uso Calcula la tangente de la *expresión*. La *expresión* está en radianes. Los cálculos se realizan con 7 dígitos significativos. La *expresión* debe estar entre + o - 200000.

Ejemplo

```
PRINT TAN(3.14/4)
1

PRINT TAN(0)
0
```

4

TIME - Interrupción

Función TIME configura y recupera el valor del temporizador de software.

Sintaxis TIME = *expresión*

variable =TIME

Vea también ONTIME, SYSTEM

Uso La declaraciónTIME se usa para recuperar o asignar un valor al temporizador de software (esto es diferente que el reloj de Calendario TIME$ respaldado por batería en tiempo real). Después de un reset, el temporizador de software es activado y el operador TIME incrementará una vez cada 5 milisegundos. El valor de TIME se expresa en segundos. Cuando el valor de TIME alcanza una cuenta de 65535,995 segundos, el valor de TIME vuelve a una cuenta de cero.

Cuando a TIME se le asigna un valor solamente, se cambia solamente la porción del número entero de TIME.

Si ss desea, la fracción de TIME se puede cambiar según lo mostrado abajo. La porción fraccionaria de TIME se separa del valor del número entero para poder hacer interrupciones periódicas de ONTIME sin ninguna pérdida en exactitud.

Ejemplo

```
> TIME=0

> PRINT1 TIME
.725

> SYSTEM(4)=0

> PRINT1 TIME
0

> SYSTEM(4)=500

> PRINT1 TIME
.5
```

TIME$ - Operador de string

Función TIME$ configura y recupera el tiempo de reloj de calendario respaldado por batería

Sintaxis TIME$ = *expresión de string*
variable de string = TIME$

Vea también DATE$

Uso Cuando está ajustado a formato correctamente, *expresión de string* configura las horas, los minutos y segundos opcionales de reloj del calendario respaldado por batería (en forma militar, 0-23). La *expresión de string* debe estar en la forma siguiente:

hh:mm:ss (por ejemplo TIME$ = "1:4:32")

La *variable de string* contiene el tiempo de reloj del calendario respaldado por batería devuelto por TIME$. TIME$ devuelve una string de longitud fija en la forma hh:mm:ss.ss.

El reloj del calendario respaldado por batería tiene una precisión +/- 1 minuto por mes con el módulo en un ambiente a 24 grados C.

Ejemplo

```
> TIME$ = "14:02:00" (configura la hora a 2:02 el P.M..)
> TIME$ = "13" (configura el reloj una hora atrasada)
> P. TIME$
13:02:33.21

> TIME$ = "0:7" (configura el reloj a 7 minutos después de  medianoche)
> 10 REM rutina para configurar el reloj una hora a más o a menos
> 20 PRINT1 "configure el reloj una hora a más o a menos"
> 21 INPUT1, "(+/-)", $(0)
> 30 $(0)=UCASE$($(0)): REM ASEGURA MAYÚSCULAS
> 40 IF $(0)="+" THEN FWD = 1: GOTO 100
> 50 IF $(0)="-" THEN FWD = 0: GOTO 100
> 60 END
> 100 HORA = VAL(TIME$)
> 110 IF FWD THEN HORA=HORA+1 ELSE HORA=HORA-1
> 120 IF HORA<0 THEN HORA=23
> 130 IF HORA>23 THEN HORA=0
> 140 TIME$ = STR$(HORA)+MID$(TIME$, 3)
```

4

TRACE - Eliminar errores

Función Mostrar el flujo de ejecución del programa y asignaciones de variables

Sintaxis TRACE *modo, número de línea*

Uso La expresión de *modo* en TRACE debe ser igual a 0, a 1 o a 2.
El *modo* 0 desactiva a la función TRACE.

El *modo* 1 exhibe el número de líneas y asignaciones de variables durante la ejecución del programa. Para cancelar el *modo* 1, entre *CTRL-C*, TRACE 0, CONT o coloque una declaración TRACE 0 en el programa.

El *modo* 2 exhibe el siguiente número de línea, cualquier asignación de variable y luego el aviso de TRACE de un paso único, "]". Para ir a la línea siguientec y parar, presione el espaciador. Para exhibir las teclas de control de un paso único de TRACE, sólo presione "H". Las teclas de control de TRACE en una etapa son:

ESPACIADOR =	paso único (igual que TRACE 2, CONT)
0 = PARAR	(igual que TRACE 0)
1 = SIN PARAR	(igual que TRACE 1, CONT
2 = CONT	(igual que TRACE 0, CONT)

Antes de entrar un número de línea en el aviso de TRACE, haga OFF a la declaración TRACE apretando "*0*". ABM Commander Plus Versions 4.11 y más nuevas automáticamente apagan al modo de TRACE de paso único siempre que usted modifique (edit) un listado de programa.

Se puede entrar cualquier declaración o COMMAND de BASIC en el aviso de TRACE de paso único. El uso típico es PRINT o asignar nuevos valores a las variables.

Usted puede especificar opcionalmente el *número de línea* de programa para comenzar a hacer TRACE. Si el *número de línea* es omitido entonces TRACE comienza en la línea corriente.

Especificar un *número de línea* para comenzar la declaración TRACE elimina la necesidad de modificar el programa para inserir declaraciones STOP.

Use la declaración TRACE 2, *número de línea* en el aviso de TRACE para reanudar la ejecución hasta que se alcanza el número de línea.

Ejemplo: Entre un comando TRACE o coloque las declaraciones TRACE en el programa para activar la función de DEBUG ON y OFF según lo requerido.

```
10 FOR I=1 TO  10
12 IF (I>4).AND.(I<7) THEN TRACE 1 ELSE TRACE  0
15 J=I*2
20 NEXT I
25 TRACE  1
30 GOTO 100
```

```
40 PRINT1 "ESTA LÍNEA NUNCA SE EJECUTA"
100 GOSUB 1000
120 END
1000 A=9999: PRINT1 "DECLARACIÓN PRINT",  A/3
1010 RETURN

> RUN

LN10 = 1
LN12
TRACE OFF
TRACE  ON
LN15 = 10
LN20 = 6
LN12
LN15 = 12
LN20 = 7
 LN12
TRACE  OFF

TRACE  ON
LN30
LN100
LN1000 = 9999
LN1000     DECLARACIÓN PRINT 3333

LN1010
LN 20
READY
> TRACE  2,15
> RUN

Step ON, press H for help
LN15 = 10
]            (Presione la barra de espacio)
LN20 = 2
]P. I
1
```

UCASE$ - Operador de string

Función UCASE$ devuelve una string que consiste en caracteres mayúsculos solamente.

Sintaxis *variable de string* = UCASE$ (*expresión de string*)

Vea también LCASE$

Uso UCASE$ devuelve una string igual a la *expresión de string* excepto que todos los caracteres alfabéticos en minúscula en la *expresión de string* se convierten en mayúsculas.

Ejemplo

```
> 10 PRINT  INPUT1 "Imprime resumen del año hasta la fecha? (S/N)", $(0)
> 20 IF UCASE$($(0))="S" THEN GOTO 100
> 30 PRINT1 UCASE$("Impresión cancelada!")
> 40 END
> 100 REM informe sumario impreso del año hasta la fecha
. . .

> RUN
Imprime resumen del año hasta la fecha? (S/N)  n
Impresión cancelada!

READY
>
```

USING - Entradas - salidas

Función Colocar formatos a valores y strings impresos con PRINT

Sintaxis PRINT USING (*formato*), *expresión*, *expresión*...

Vea también PRINT, CR, SPC, TAB, @(*Línea*,*columna*)

Uso El *formato* especifica cómo será impresa la lista de *expresiones*.

Pueden ser especificados tres formatos diferentes. PRINT configura el *formato* del punto decimal (el punto, no la coma, como es la norma en Estados Unidos) para notación exponencial o decimal y se usa para imprimir una columna de números con todas los puntos alineados. La impresión del *formato* de string se usa para imprimir una string de longitud fija (incluyendo caracteres delimitadores de string NULL y **retorno de carro**).

Una vez que se especifica un *formato* PRINT, será usado en todas las declaraciones subsecuentes de la declaración PRINT o hasta que se encuentra una PRINT USING(0).

La declaración PRINT USING(0) hace que sean mostrados números en el rango de ±0,2 al ± 99999999 en la notación decimal. Los números fuera de este rango serán exhibidos en la notación exponencial.

Pueden aparecer varias declaraciones USING en una sola declaración PRINT.

Use una coma después de la declaración USING para prevenir el mensaje de error BAD SYNTAX.

Colocando formato a números

Sintaxis PRINT USING (#.#), *expresión numérica*

uso PRINT USING (#.#) hará que el valor de todas las *expresiones numéricas* subsecuentes sean impresas con un número fijo de dígitos antes y después la coma. La cantidad de caracteres de signo libra, "#", antes y después del punto determinan el número de dígitos significativos que serán impresos.

La coma puede ser omitida si se desea mostrar solamente un número entero.

La cantidad máxima de caracteres "#" es ocho en total. Si el valor que se mostrará no cabe en el formato actualmente especificado entonces BASIC mostrará un carácter de signo de interrogación (?) seguido por el valor en el formato USING(0).

Ejemplo

```
> 10 PRINT1 USING (##.###),
> 20 FOR I=0 PRINT1 TO 30 STEP 5
> 30 SQR(I**3)
> 40 NEXT I
> RUN
0.
11.180
31.622
58.094
89.442
?125
?164.31677
```

Colocando formato a números exponenciales

Sintaxis PRINT USING (Fx), *expresión numérica*

Uso Esta función del formato PRINT hará que las *expresiones numéricas* subsecuentes sean mostradas en un formato exponencial fijo de notación de coma flotante. El valor *x* determina cuántos dígitos significativos serán impresos. El valor mínimo de x es 3 y el valor máximo es 8.

Ejemplo

```
> 10 PRINT1 USANDO (F3),
> 15 FOR J=1 TO  2
> 20 FOR K=1 TO  5
> 30 PRINT1 J*K, SPC(2),
> 40 NEXTS K
> 45 PRINT1
> 50 J NEXT
1.00 EO 2.00 EO 3.00 EO 4.00 EO 5.00 EO
2.00 EO 4.00 EO 6.00 EO 8.00 EO 1.00 E1
> P. 7, SPC(2), U.(F4), 4, SPC(2), U.(F3), 5, SPC(2), USING(F8), 0
7.00 EO 4.000 EO 5.00 EO 0.0000000 EO
```

Formato de strings

Sintaxis PRINT USING(\ *expr* *), expresión de string*

Uso El valor *expr* es la cantidad de caracteres en la *expresión de string* que serán impresos. PRINT comienza siempre al principio del string. Esta función se puede usar para imprimir los strings que contienen los caracteres **null** (ASCII 0) y **retorno de carro** (ASCII 13).

Ejemplo

```
10 REM asigna el espacio para 10 strings
20 REM 254 caracteres cada string máximo
30 REM STRING (254+1)*10+1,254
40 STRING 2551,254
50 $(0)="0123456789"
60 PRINT1 $(0)
65 PRINT1 "Imprima una porción de un string"
70 PRINT1 USING(\5\), $(0)
75 $(0)="ABC"+CHR$(10)
77 PRINT1 "BASIC marca el fin de string con un  ",
78 PRINT1 ASC($(0), LEN($(0))+1)
80 PRINT1 $(0)
83 PRINT1 "Imprime más allá del fin del marcador del string"
85 L=LEN($(0))*4
90 PRINT1 USING(\L\), $(0)
100 ASC($(0), 1)=0
105 PRINT1 "Una impresión normal no muestra un carácter null"
110 PRINT1 $(0)
120 PRINT1 "PRINT con formato de string lo hará!",
121 PRINT1 USING (\3\), $(0)

> RUN
0123456789
Imprima una porción de un string
01234
BASIC marca el fin de string con un  13
ABC
Imprime más allá del fin del marcador del string
ABC
56789
Una impresión normal no muestra un carácter null

PRINT con formato de string lo hará! BC
```

VAL - Operador de string

Función VAL devuelve el equivalente numérico de una expresión de string

Sintaxis *variable de string* = VAL (*expresión de string*)

Vea también STR$

Uso VAL convierte la *expresión de string* en un número equivalente. Si la *expresión de string* contiene caracteres no numéricos, entonces VAL vuelve el número hasta el cuando haya un carácter no numérico. Si la *expresión de string* comienza con un carácter no numérico entonces VAL retorna un 0.

```
10 INPUT1 "Entre por favor la referencia 1", $(0)
20 IF (VAL($(0))>=0).AND.(VAL($(0))<4096) GOTO 50
30 PRINT1 "La referencia debe estar en el rango 0 a 4095"
40 GOTO 10
50 SP1=VAL($(0))
10 $(0)="M1@X23.4Y6.8Z1.2 ="
20 XPOS=INSTR($(0), "X")+1
30 YPOS=INSTR($(0), "Y")+1
40 ZPOS=INSTR($(0), "Z")+1
50 X=VAL(MID$($(0), XPOS))
60 Y=VAL(MID$($(0), YPOS))
70 Z=VAL(MID$($(0), ZPOS))
80 PRINT1 X*Y+Z, "=", 23.4*6.8+1.2

READY
> RUN
160.23 = 160.23

READY
>
```

WORD - Operador avanzado

Función WORD lee desde o escribe dos bytes a una posición de memoria específica

Sintaxis *var* = WORD (*dirección*)
WORD (*dirección*) = *expr*

Vea también BYTE, LOAD@, STORE @

Uso La *dirección* es una expresión en el rango de 0 a 65535, representando una posición de memoria de dos bytes. WORD recupera o asigna un valor de número entero (0 a 65535). WORD se puede usar para almacenar valores de números enteros en una región de memoria protegida BASIC (de MTOP a 32767). WORD se puede también usar para recuperar valores de números enteros en cualquier lugar en la memoria.

Ejemplo Almacenar valores en memoria permanente después de archivar programas en el banco 1.

```
1000 MEMORIA1 = SYSTEM(6): REM Primera localización libre
1010 FOR IDX = 1 TO  100
1020 WORD (MEMORIA1+I) = REG(I)
1030 NEXT IDX
```

Recupera los valores almacenados en memoria permanente después de archivar programas en el banco 1

```
1000 MEMORIA2 = SYSTEM(6): REM  primera localización libre
1010 FOR IDX = 1 TO 100
1020 REG(I) = WORD(MEMORIA2+I)
1030  NEXT IDX
```

4

@(línea, columna) - Entradas - salidas

Función Colocación de cursor usando secuencias de escape ANSI

Sintaxis @ (*línea*, *columna*)

Vea también PRINT, CR, SPC, TAB, USING

Uso Este operador se usa en declaraciones PRINT para generar una secuencia de escape requerida para colocar el cursor en un terminal compatible ANSI o de la DEC VT100. La *línea* especifica la posición vertical y la *columna* especifica la posición horizontal respecto a la pantalla. Este operador de colocación del cursor se usa a menudo para poner fácilmente el texto en un terminal de interface de operador.

Ejemplo1 Las dos declaraciones PRINT siguientes son equivalentes.

```
LÍNEA = 5: COLUMNA = 50
PRINT1 @(LÍNEA,COLUMNA), "DERECHA superior"

PRINT1 CHR$(27), "[",LÍNEA, ";",COLUMNA, "H", "DERECHA superior"
```

Ejemplo 2 Las declaraciones PRINT siguientes de control de cursor dan al usuario más control sobre la pantalla del ABM Commander Plus.

```
PRINT1 @(1,1); : REM coloca el cursor en las coordenadas 1,1
PRINT1 CHR$(27), "[2J"; : REM Borre la Pantalla
PRINT1 CHR$(27), "[2L"; : REM Apague el Cursor
PRINT1 CHR$(27), "[2K"; : REM  Enciende el Cursor
```

OPERADORES MATEMÁTICOS



En este capítulo:

Tabla de operadores matemáticos

Los operadores matemáticos descritos en este capítulo son:

+
-
*
/
**

OPERADOR	DESCRIPCIÓN	FORMA GENERALIZADA	EJEMPLO
+	SUMA	expresión + expresión	PRINT 2+3 5
-	Substracción	expresión - expresión	PRINT 2-3 -1
*	Multiplicación	expresión * expresión	PRINT 2*3 6
/	División	expresión / expresión	PRINT 2/3 .66666667
**	Elevación a potencia	expresión ** expresión	PRINT 2**3 8

OPERADORES LÓGICOS Y DE COMPARACIÓN



En este capítulo:

Tabla de operadores matemáticos

Los operadores lógicos y de comparación descritos en este capítulo son:

LÓGICOS		**DE COMPARACIÓN**		
AND	**.OR.**			
XOR.	**NOT()**	**< >**	**<**	**<=**

OPERADORES LÓGICOS

Los operadores lógicos realizan sus funciones en números enteros válidos en bit a bit (16 bits). Los argumentos de números que no son enteros en el rango 0 a 65535 (0FFFFH) inclusive son truncados. Números fuera de este rango generarán el mensaje, ERROR: BAD ARGUMENT.

OPERADOR	DESCRIPCIÓN	FORMA GENERALIZADA	EJEMPLO
.AND.	AND	expresión**.AND.**expresión	PRINT 2.AND.3 2
.OR.	OR	expresión**.OR.** expresión	PRINT 2.OR.3 3
.X0R.	OR EXCLUSIVO	expresión **.XOR.** expresión	PRINT 2.XOR.3 1
NOT()	NOT	**NOT**(expresión)	PRINT NOT(2) 65533

TABLAS DE VERDAD DE LOS OPERADORES LÓGICOS

A .AND. B			A. OR. B			A. XOR.B			NOT (A)	
A	B	RESULTADO	A	B	RESULTADO	A	B	RESULTADO	A	RESULTADO
0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	1	0
1	0	0	1	0	1	1	0	1		
1	1	1	1	1	1	1	1	0		

OPERADORES DE COMPARACIÓN

Los operadores de comparación se usan para probar si una relación especificada entre dos expresiones es VERDAD o FALSA. Si la relación es VERDADERA entonces se da como resultado todos unos, 65535 (0FFFFH). Si la relación es FALSA entonces da como resultado un 0.

Tabla de operadores de comparación

OPERADOR	DESCRIPCIÓN	FORMA GENERALIZADA	EJEMPLO
=	IGUAL A	expresión **=** expresión	PRINT 2=3 0
<>	NO IGUAL A	expresión**<>** expresión	PRINT 2<>3 65535
>	MAYOR QUE	expresión **>** expresión	PRINT 2>3 0
<	MENOR QUE	expresión **<** expresión	PRINT 2<3 65535
>=	MAYOR O IGUAL QUE	expresión **>=** expresión	PRINT 2>=3 0
<=	MENOR O IGUAL QUE	expresión **<=** expresión	PRINT 2<=3 65535

Puesto que los operadores de comparación entregan como resultado un número entero válido, específicamente un 0 o 65535, entonces los operadores lógicos pueden usar los resultados de operaciones de comparación para formar expresiones de comparación complejas.

Ejemplos

```
> 10 IF  X<=Y.AND.(X>Z.OR.X=O) THEN...

> 10 IF NOT(A.OR.B).AND.C THEN...
```

En el primer ejemplo de arriba, fueron utilizados paréntesis para causar que el resultado de la operación OR lógico sea usado como uno de los argumentos en la operación AND lógica. A menudo, las expresiones complejas se pueden escribir con pocos paréntesis si el usuario entiende la precedencia de operadores. Vea el CAPÍTULO 2 para repasar la precedencia de operadores en expresiones. Cuando tenga dudaS sobre precedencia de operadores, se recomienda que sean usados paréntesis.

MENSAJES DE ERROR



En este capítulo:

Los mensajes de error descritos en este capítulo son:

Cuando ocurren errores en el modo COMMAND, será generado e impreso un mensaje de error en el puerto de comando. Cuando un error ocurre durante la ejecución de programa, se termina el programa, se genera y se imprime un mensaje de error en el puerto de comando. Luego el número de línea de programa que causó el error imprime en el puerto del comando una ' X ' donde aproximadamente ocurrió el error en la línea.

Ejemplo

```
ERROR: BAD SYNTAX - IN LINE 110
110 PRINT 14+12 *
----------------x
```

ARGUMENT STACK OVERFLOW

Un error de DESBORDAMIENTO DE "STACK" DEL ARGUMENTO ocurre normalmente cuando se hace una tentativa de hacer POP de datos del "stack" cuando no hay datos presentes. Este error ocurrirá también si el usuario desborda el "stack" del argumento empujando también muchas expresiones sobre el "stack".

ARITHMETIC OVERFLOW

Si el resultado de una operación aritmética excede el límite superior de un número de coma flotante en BASIC, ocurrirá un ERROR ARITMÉTICO DE DESBORDAMIENTO. El número más grande de coma flotante BASIC es +/- 99999999E+127. Por ejemplo, 1E+70*1E+70 causaría un error ARITMÉTICO DE DESBORDAMIENTO.

ARITHMETIC UNDERFLOW

Si el resultado de una operación aritmética excede el límite más bajo de un número de coma flotante de BASIC, ocurrirá un ERROR ARITMÉTICO DE DESBORDAMIENTO DE CAPACIDAD INFERIOR. El número más pequeño de coma flotante BASIC es +/- 1E-127. Por ejemplo, 1E-80/1E+80 causaría un error ARITMÉTICO DE DESBORDAMIENTO DE CAPACIDAD INFERIOR.

ARRAY SIZE -SUBSCRIPT OUT OF RANGE

(TAMAÑO DEL ARREGLO - SUBÍNDICE FUERA DE RANGO) Si un arreglo tiene declarada la dimensión por una declaración DIM y luego usted trata de tener acceso a una variable que esté afuera de los límites de la dimensión, será generado un error de TAMAÑO DEL ARREGLO. Este error también ocurrirá si usted trata de hacer una nueva dimensión de un arreglo.

Ejemplo

```
> DIM A(10)
> PRINT A(11)
ERROR: ARRAY SIZE - SUBSCRIPT OUT OF  RANGE
READY
>
```

BAD ARGUMENT (MAL ARGUMENTO)

Cuando el argumento de un operador no está dentro de los límites del operador será generado un ERROR DE ARGUMENTO. Por ejemplo, A=TRANSFER(257) generaría un error de BAD ARGUMENT porque el argumento para la declaración TRANSFER se limita al rango 0 a 255. PRINT ASC($(2), 1) generará un error si el espacio de almacenaje de string no ha sido asignado por STRING. (TRANSFER solo existe en algunos módulos)

BAD SYNTAX (MALA SINTAXIS)

Un error BAD SYNTAX significa que fue entrado un comando, declaración u operador BASIC inválido y BASIC no puede interpretar la entrada. El usuario debe comprobar y cerciorarse de que todo fue tecleado correctamente. Un error de BAD SYNTAX puede también ser generado si una palabra clave reservada se usa como parte de una variable (vea el apéndice C).

CAN'T CONTINUE (NO PUEDE CONTINUAR)

La ejecución de programa puede ser parada entrando un < *CTRL-C* > a través del puerto del comando o ejecutando una declaración STOP. Normalmente, la ejecución de programa puede ser reanudada tecleando el comando CONT. Sin embargo, si el usuario corrige el programa después de que pare la ejecución y después entra el comando CONT, será generado un error CAN'T CONTINUE. Se debe teclear < *CTRL-C* > durante la ejecución de programa o debe ser ejecutada una declaración STOP antes de que el comando CONT trabaje.

CONTROL "STACK" OVERFLOW

Los errores de DESBORDAMIENTO DE "STACK" DE CONTROL ocurrirán normalmente si se ejecuta un RETURN antes de un GOSUB, de un WHILE o UNTIL, antes de que DO o de un NEXT antes de un FOR.

Un error de DESBORDAMIENTO DE "STACK" de CONTROL también ocurrirá si el puntero del stack de control es forzado "fuera de los límites". 158 bytes de memoria se asignan para el "stack" del control.

Los lazos FOR-NEXTs requieren 17 bytes de "stack" de control-DO-UNTIL, DO-WHILE, y GOSUB requieren 3 bytes de "stack" de control. Esto significa que 9 lazos FOR-NEXT uno dentro de otro es el máximo que BASICpueden manejar porque 9 por 17 igual a 153. Si el usuario trata de usar más "stack" de control que el disponible, será generado un error de DESBORDAMIENTO DE "STACK" DE CONTROL .

CORRUPTED PROGRAM ENCOUNTERED

Cuando se encuentra un programa corrupto en la memoria almacenada del programa entonces el END del marcador del fin de archivo se mueve a la primera línea de programa válida antes de la corrupción. Esto trunca el resto del programa y suprime todos los programas que lo si guen. La memoria del programa podría posiblemente ser cambiada debido al ruido eléctromagnético tal como electricidad estática.

DIVIDE BY ZERO

Si se trata de hacer una división por CERO (por ejemplo, 12/0), ocurrirá el error DIVIDE BY ZERO .

EXPRESSION TOO COMPLEX (EXPRESIÓN MUY COMPLEJA)

Ocurre un error EXPRESIÓN MUY COMPLEJA cuando BASIC no tiene bastante espacio del "stack" para evaluar una expresión (también muchos paréntesis). Nunca hemos visto este error en el mundo real, sin embargo, si usted maneja generar este mensaje entonces la expresión debe ser simplificada obteniendo resultados intermedios.

INVALID LINE NUMBER (NÚMERO DE LÍNEA INVÁLIDA)

Este error ocurre normalmente cuando el programa trata de ir a un número de línea que no existe. El error se podría causar por cualesquiera de las declaraciones que usan nómerode línea, por ejemplo GOTO, GOSUB, ONPORT y otros. El error puede también ocurrir cuando el programa en el buffer de EDIT (PROGRAM=0) se corrompe.

Para verificar al programa cero, entre lo siguiente:

> PRM 0 seleccionar el programa cero

> P. LOF Longitud de PRINT del programa

SI LOF devuelve la cantidad de bytes de la longitud del programa entonces el programa cero no está corrupto.

SI LOF genera el mensaje INVALID LINE NUMBER entoces algo ha cambiado el contenido del programa cero. Para corregir el error, genere un comando NEW y recargue una copia de respaldo del programa.

MEMORY ALLOCATION (ASIGNACIÓN DE MEMORIA)

Se generan los errores de ASIGNACIÓN DE MEMORIA cuando el usuario trata de tener acceso a los strings que son 'está fuera de' los límites definidos de string o cuando hay memoria escasa para el almacenaje variable. Además, si la parte superior del valor de la memoria, MTOP, se asigna un valor que no contiene ninguna memoria de datos, ocurrirá un error de asignación de memoria.

NO DATA (SIN DATOS)

Si se ejecuta una declaración READ y no existe ninguna declaración de DATOS o se han leído todos los datos y una instrucción del RESTORE no fue ejecutada será generado el mensaje de error : NO DATA-IN LINE XXX.

NO ENOUGH FREE SPACE

Se genera el mensaje NO ENOUGH FREE SPACE después de un COMANDO SAVE cuando la longitud del programa actualmente seleccionado (generalmente programa 0 en el BUFFER de EDIT) excede el número de bytes restantes en la memoria almacenada del programa.

El número de programas almacenados y el número de bytes restantes se exhibe después de reset cuando es el AUTOSTART está dentro del modo EDIT (modo 0 o 3).

El espacio disponible libre de almacenaje del programa también es determinado entrando un comando DELPRM para un programa que no exista.

Para determinar la longitud del programa que está siendo archivado use LOF.

Ejemplo

> DELPRM 5

4 stores program, 6381 bytes free

> P. LOF 7673

PROGRAM ACCESS

Al tratar de seleccionar un programa almacenado que no exista se generará el mensaje de error PROGRAM ACESS. El número del último programa almacenado se exhibe después de un reset cuando es el AUTOSTART está dentro del modo EDIT (modo 0 o 3).

STRING TOO LONG

El mensaje STRING DEMASI ADO LARGO se genera cuando se trata de crear un STRING más largo que la longitud máxima de string definida por la declaración STRING. Use la STRING para asignar la memoria para strings más largos o para romper la declaración STRING en segmentos.

UNABLE TO VERIFY

Si ocurre un error mientras que se está archivando un programa, será generado un error INCAPAZ DE VERIFICAR .

AVANZADO



En este capítulo:

Formato de almacenamiento de coma flotante

Las declaraciones STORE @ y LOAD @ se puede usar en un programa BASIC para archivar y para recuperar números de coma flotante en posiciones de memoria absolutas. Cada número de coma flotante requiere seis bytes de memoria para el almacenaje. Todas las variables sin dimensión se almacenan como números de coma flotante en un formato BCD según lo mostrado en el ejemplo siguiente.

Ejemplo

> PUSH 1.2345678

> STORE @ 30000

Localización	Valor	Descripción
30000	81H	Exponente
		$-7F_H$ = 10**-1
		80_H = 10**0
		81_H = 10**1
		82_H = 10**2
		Número cero = exponente cero
29999	00H	Bit de signo - 00H = positivo, 01H = negativo
29998	78H	Los dos dígitos menos significativos BCD
29997	56H	Los dos dígitos menos significativos siguientes BCD
29996	34H	Los dos dígitos más significativos siguientes BCD
29995	12H	Los dos dígitos más significativos BCD

Formato de almacenamiento de variables sin dimensión

Las variables requieren 8 bytes de memoria para el almacenaje. Se usan dos bytes para describir el nombre de la variable mientras que se usan 6 bytes restantes para almacenar el número de coma flotante según lo descrito previamente. El ejemplo siguiente muestra cómo sería almacenada la variable CHAR .

Ejemplo

```
> STRING 0.0

> CHAR = 12345

> P. WORD(104H)

32767
```

Localización	Valor	Descripción
32766	52H	El valor ASCII para el último carácter usado para definir una variable. En este ejemplo, el valor ASCII para el carácter "R".
32765	119	El valor ASCII para el primer carácter usado para definir una variable más 26 veces la cantidad de caracteres en el nombre de la variable que mayor que 2 (67 + 26 * (4-2) = 119).
32764	80H	Exponente de la coma flotante
32763	0	Bit de signo
32762	0	Los dos dígitos menos significativos BCD
32761	50H	Los dos dígitos menos significativos siguientes BCD
32760	34H	Los dos dígitos más significativos siguientes BCD
32759	12H	Los dos dígitos más significativos BCD

Formato de almacenamiento de variables con dimensión

Las variables con dimensión requieren 8 bytes de memoria para el almacenaje. El ejemplo siguiente muestra como sería almacenada la variable ARRAY(2)

Ejemplo

```
> STRING 0.0

> ARRAY(2) = -12.8

> P. WORD(104H)


32767
```

Localización	Valor	Descripción
32766	217	El valor ASCII para el último carácter usado para definír el nombre de la variable de arreglo más 128. En este ejemplo, el valor ASCII para el carácter "Y" (89 +128 = 217).
32765	143	El valor ASCII para el primer carácter usado para definír una variable más 26 vecesd la cantidad de caracteres en el nombre de la variable mayor que 2 (65 + 26 * (5-2) = 143).
32764	11	Cantidad máxima de elementos en la variable con dimensión. Por defecto éste es 11 (ARRAY(0) hasta el ARRAY(10)).
32763	6	El byte menos significativo de la dirección base para el ARREGLO
32762	4	El byte más significativo de la dirección base para el ARREGLO

Formato de almacenamiento de variables string

La declaración de string define la longitud máxima de string y la memoria asignadas para el almacenaje del string. La STRING 2551, 254 asigna la memoria para 10, 254 cadenas de caracteres (10 * (254+1) + 1 = 2551). Las variables de string se almacenan a partir de WORD(104H) hasta MTOP según lo mostrado en el ejemplo siguiente.

Ejemplo

> la STRING 21,9 asigna la memoria para dos strings de nueve caracteres > $(0)="UNO"

> $(1)="DOS"

> PRINT CHR$(BYTE(WORD(104H)))

U

> PRINT CHR$(BYTE(1+WORD(104H)+9*1))

D

8

Comunicaciones con CRC-16 automático

El control por redundancia cíclico (CRC) es un medio confiable de verificación para saber si hay errores de comunicación. Un algoritmo de CRC es mucho más eficaz que los algoritmos de paridad y verificación de suma (Checksum). BASIC ampliado de FACTS Eng. utiliza un CRC de 16 bits y se refiere así como CRC-16. Esta implementación del CRC se puede configurar para comunicarse con otros dispositivos que usen un CRC de 16 bits, tal como el protocolo Modbus RTU.

Operación del CRC

El dispositivo que transmite genera dos caracteres CRC-16 para cada transmisión y agrega esos caracteres al fin de la transmisión. El dispositivo de recepción después calcula el CRC-16 en los datos entrantes y verifica que el resultado es igual que los caracteres reales CRC-16 recibidos. La función CRC-16 requiere que los dispositivos de transmición y de recepción usen el mismo algoritmo CRC, usen el mismo resto inicial, y transmitan los caracteres del CRC en la misma orden. La función del CRC se habilita y se deshabilita según lo mostrado abajo.

10 SYSTEM(2)=NOT(0): REM habilita la función del CRC

10 SYSTEM(2)=0 del CRC: REM deshabilita la función del CRC

Transmitiendo con CRC

Cuando se habilita la función CRC, BASIC calcula dos caracteres CRC-16 para cada declaración PRINT. Cada carácter transmitido por cualquier declaración PRINT se incluye en el cálculo. Por defecto, BASIC agrega los dos caracteres CRC-16, primero el byte más significativo (MSB) y al último el byte menos significativo (LSB) al fin de la transmisión. La orden en la cual se agregan los caracteres CRC-16 en el fin de la transmisión se puede seleccionar por el programa de usuario según lo mostrado abajo.

10 SYSTEM(3)=0: REM Tx MSB primero, LSB al último

10 SYSTEM(3)=NOT(0): REM Tx LSB primero, MSB al último

Recepción con el CRC

Cuando se habilita la función CRC, BASIC calcula dos caracteres CRC-16 para cada declaración INPUT. Cada carácter recibido por cualquier una declaración INPUT se incluye en el cálculo. Por defecto, BASIC mira los dos últimos caracteres INPUT para los dos caracteres CRC-16, MSB primero y LSB al último. BASIC busca los caracteres CRC-16 en la misma orden en la cual BASIC transmitiría los caracteres. Si los caracteres no son iguales, ha ocurrido un error de comunicación. BASIC configura un bit de error de comunicación, COMERR, todos 1s cuando hay un error de CRC. Si se reciben caracteres correctos CRC-16, BASIC configura COMERR a 0.

Resto inicial

El resto inicial (los caracteres iniciales de CRC) son todos 1s después de un reset. El resto inicial se puede cambiar por el programa de usuario según lo mostrado abajo.

10 WORD(132H) = 0:REM cambia resto inicial a todos 0s

10 WORD(132H) = 0FFFFH: REM cambia resto inicial a todos 1s

Examinando los caracteres CRC-16

Para los propósitos de eliminación de errores, los dos últimos caracteres CRC-16 generados ya sea para una transmisión o una recepción puede ser examinados por BASIC según lo mostrado abajo.

```
10 PRINT1 "el valor del ASCII del último  MSB de CRC-16 =",
15 PRINT1 PICK(SYSTEM(5), H)
20 PRINT1 "el valor del ASCII del último  LSB de CRC-16 = ",
25 PRINT1 PICK(SYSTEM(5), l)
```

Programa de demostración del CRC

El programa siguiente muestra el uso de la función del CRC ya incluída en BASIC. El programa permite que usted simule a recepción de datos usando el teclado. En la práctica, los datos recibidos en una STRING por la declaración INPUT serían examinados usando al operador ASC. El operador ASC puede ser usados para "agarrar" caracteres en un string.

```
100 REM Versión de programa  CRC-16 para el BASIC ampliado de FEAVCTS Eng
110 CL:EAR : STRING 8001,79
120  PRINT1 "Resto inicial CRC-16 por defecto =", WORD(132H)
130 SETINPUT 0,1,0,6,65535,5000
140 PRINT1 "CRC-16 CON MSB PRIMERO"
150 PRINT1 "Entre por favor lo siguiente: "
160 SYSTEM(2)=1: SYSTEM(3)=0: REM PERMITE CRC-16 MSB PRIMERO
170 GOSUB 240
180 PRINT1 "CRC-16 CON LSB PRIMERO"
190 PRINT1 "Entre en por favor lo siguiente: "
200 SYSTEM(2)=1: SYSTEM(3)=1: REM ACTIVA CRC-16 LSB PRIMER0
210 SETINPUT 0,1,0,6,65535,5000
220 GOSUB 240
230 END
240 PRINT1 "KJHS",
250 INPUT1, $(1)
260 SYSTEM(2)=0: REM Deshabilita verificación de fallas(error checking) CRC-16
270 IF  COMERR THEN GOSUB 290 GOSUB ELSE 300
280 RETURN
290 PRINT1 "ERROR DE CRC-16"
300 PRINT1: PRINT1
310 PRINT1 "CRC-16 MSB - >", PICK(SYSTEM(5), H),
```

```
315 PRINT1 "CARÁCTER - >", CHR$(PICK(SYSTEM(5), H))
320 PRINT1 "CRC-16 LSB - >", PICK(SYSTEM(5), L),
325 PRINT1 "CARÁCTER - >", CHR$(PICK(SYSTEM(5), L))
330 PRINT1: PRINT1
340 RETURN
```

COMO FINALIZAR UN PROGRAMA



En este apéndice:

Colocando el módulo CoProcessor en funcionamiento de la mejor forma

A

Después de que se haya terminado la programación y el ciclo de afinamiento del programa (algunos llaman a esto Debugging), el módulo está listo para ser puesto en servicio a largo plazo.

Se recomiendan los siguientes pasos para tener una máxima confiabilidad como se requiere en la mayoría de los usos industriales. Estos pasos ayudarán a prevenir una operación indeseable debido a un error no detectado en el programa, señales exteriores tales como descargas electrostáticas o señales incontrolables debido a ambientes fuera de tolerancia.

1. Haga una copia de respaldo del programa o programas.
2. Deshabilite la acción de BREAK o < *CTRL-C* > agregando una declaración "BREAK=0" al programa o a los programas.
3. Fuerce la ejecución dle programa agregando una declaración “LOCKOUT=NOT(0)" al programa.
4. Entre el comando apropiado de AUTOSTART.

Para impedir que haya más modificaciones (EDIT) en la ejecución del programa, vea la declaración BREAK.

También, el LOCKOUT puede ser desactivado y se puede permitir BREAK bajo control de software. Vea la descripción de la declaración END para un ejemplo.



PALABRAS RESERVADAS



En este apéndice:

Palabras reservadas

Lo que sigue es la lista alfabética de todas las palabras reservadas para uso por el intérprete BASIC ampliado de FACTS Eng. Aunque no se utilizan todas las palabras claves enumeradas en el sistema de instrucciónes del módulo ASCII/BASIC, las variables pueden no contener cualquiera de las palabras mostradas

Palabras reservadas

ATN
AUTOSTART
BIT
BYTE
BREAK
CALL
CBY
CLEAR
CLOCK
COMERR
CONT
COS
CR
DATA
DBY
DELAY
DELPRM
DIM
DO
DSR
DTR
EDIT
END
ERASE
EXP
FOR
GOPRM
GO_PROGRAM
GOSUB
GOTO
IDLE
IF
INLEN
INPLEN
INPUT
INSTR
INT
LEN
LET
LOCKOUT
LOF
LOG
MTOP
NEW
NEXT
NOT
EN
ONERR
ONEX1
ONPORT
ONTIME
PGM
PI
PRINT
PRM
PROGRAM
POP
PUSH
RAM
READ
REM
RENUMBER
RESET
RESTORE
RETI
RETURN
RND ROM
RROM
SAVE
SETINPUT
SETPORT
SGN
SIN
SPC
SQR
STEP
STOP
STRING
TAB
TAN
THEN
TIME
TO
UNTIL
VAL
WHILE
WORD
XBY
XFER

Símbolos reservados

@	**$**	**,**	**.**		
()	**+**	**-**	*****	**/**	******
>	**< =**	**>**	**= <**	**=**	**< >**
AND.	**OR.**	**XOR.**			

TABLAS DE CARACTERES ASCII



En este apéndice:

TABLA DE CARÁCTERES DE CONTROL

Código de control	Carácter ASCII	Valor Decimal	Definición Abreviatura
< Ctrl @ >	NULL	0	NULL
< Ctrl A >	SOH	1	Comienzo del título
< Ctrl B >	STX	2	Comienzo del texto
< Ctrl C >	ETX	3	END del texto
< Ctrl D >	EOT	4	END de la transmisión
< Ctrl E >	ENQ	5	Investigación
< Ctrl F >	ACK	6	Reconocer
< Ctrl G >	BELL	7	Campana
< Ctrl H >	BS	8	Tecla de retroceso
< Ctrl I >	HT	9	Tabulación horizontal
< Ctrl J >	Lf	10	Avance de línea
< Ctrl K >	VT	11	Tabulación vertical
< Ctrl L >	FF	12	(Form Feed) Alimentación de carro
< Ctrl M >	TAN	13	Retorno de carro
< Ctrl N >	SI	14	(Shift out) Cambio hacia fuera
< Ctrl O >	DLE	15	cambiar de puesto adentro
< Ctrl P >	DC1	16	Escape de trasmisión de datos
< Ctrl Q >	DC2	17	Control de dispositivo 1
< Ctrl R >	DC3	18	Control de dispositivo 2
< Ctrl S >	DC4	19	Control de dispositivo 3
< Ctrl T >	NAK	20	Control de dispositivo 4
< Ctrl U >	SYN	21	Reconocimiento negativo
< Ctrl V >	ETB	22	IDLE síncrona
< Ctrl W >	CAN	23	END del bloque de transmisión
< Ctrl X >	EM	24	Cancelación
< Ctrl Y >	SUB	25	Final del medio
< Ctrl Z >	ESC	26	Substituto
< Ctrl [>	FS	27	Escape
< Ctrl \ >	GS	28	Separador de archivo
< Ctrl] >	RS	29	Separador del grupo
< Ctrl ^ >	US	30	Separador de registro
< Ctrl _ >	SP	31	Separador de unidad
	DEL		Espacio
			Borrar

C

TABLA DE CONVERSIÓN ASCII

Chr	Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex
NULL	0	0	SP	32	20	@	64	40	`	96	60
SOH	1	1	!	33	21	A	65	41	a	97	61
STX	2	2	"	34	22	B	66	42	b	98	62
ETX	3	3	#	35	23	C	67	43	c	99	63
EOT	4	4	$	36	24	D	68	44	d	100	64
ENQ	5	5	%	37	25	E	69	45	e	101	65
ACK	6	6	&	38	26	F	70	46	f	102	66
BEL	7	7	'	39	27	G	71	47	g	103	67
BS	8	8	(	40	28	H	72	48	h	104	68
HT	9	9	)	41	29	I	73	49	i	105	69
LF	10	A	*	42	2A	J	74	4A	j	106	6A
VT	11	B	+	43	2B	K	75	4B	k	107	6B
FF	12	C	,	44	2C	L	76	4C	l	108	6C
CR	13	D	-	45	2D	M	77	4D	m	109	6D
SO	14	E	.	46	2E	N	78	4E	n	110	6E
SI	15	F	/	47	2F	P	79	4F	o	111	6F
DLE	16	10	0	48	30	Q	80	50	p	112	70
XON	17	11	1	49	31	R	81	51	q	113	71
DC2	18	12	2	50	32	S	82	52	r	114	72
XOFF	19	13	3	51	33	T	83	53	s	115	73
DC4	20	14	4	52	34	U	84	54	t	116	74
NAK	21	15	5	53	35	V	85	55	u	117	75
SYN	22	16	6	54	36	W	86	56	v	118	76
ETB	23	17	7	55	37	X	87	57	w	119	77
CAN	24	18	8	56	38	Y	88	58	x	120	78
EM	25	19	9	57	39	Z	89	59	y	121	79
SUB	26	1A	:	58	3A	[	90	5A	z	122	7A
ESC	27	1B	;	59	3B	\	91	5B	{	123	7B
FS	28	1C	<	60	3C	]	92	5C	|	124	7C
GS	29	1D	=	61	3D	^	93	5D	}	125	7D
RS	30	1E	>	62	3E	_	94	5E	~	126	7E
US	31	1F	?	63	3F		95	5F	DEL	127	7F

Los caracteres 128 hasta 255 dependen de cada aparato y es por eso que no son listados en este apéndice.



VELOCIDAD DE EJECUCIÓN DE PROGRAMAS BASIC

En este apéndice:

Se pretende que este apéndice suministre al usuario algunas ideas de como determinar la velocidad de ejecución del programa BASIC ampliado de FACTS Eng. Debido a la cantidad de posibilidades de programación, sería impráctico proporcionar una lista de declaraciones de BASIC y los tiempos de ejecución. Sin embargo, serán mostradas velocidades típicas para ejecutar tareas comunes. Finalmente, se presentan algunas sugerencias para programación para aquellas aplicaciones donde es importante una ejecución a la velocidad máxima.

BASIC ampliado de FACTS ENGINEERING es un BASIC interpretativo completo altamente eficiente. La eficacia de la puesta en práctica es ejemplificada por el hecho que la característica conducida interrupción del contador de tiempo consume solamente el 0,4% del tiempo total de la CPU.

Se puede fácilmente medir el tiempo de la velocidad de ejecución de una sección de un programa BASIC usando el temporizador de los módulos.

```
5000 TIME=0: REM Hora actual reseteada en segundos
5020 SYSTEM(4)=0: REM Resetea la hora corriente en milisegundos
5030 REM Comienza el cálculo del tiempo
5040 FOR I=1 TO 1000
5050 GOSUB 100: REM Mide el tiempo de ejecución de la subrutina en la línea 100
5060 NEXT I : REM  Termina la medición de tiempo
5070 T=TIME: Rem Guarda el tiempo requerido para 1000 ejecuciones
5080 PRINT "Tiempo requerido para ejecutar la subrutina es", T-1.655,
5090 PRINT "milisegundos"
```

Ejemplo

```
> 50 GOTO 5000
> 100 A = 127 * 2
> 110 RETURN
> RUN
El tiempo requerido para ejecutar la subrutina es 1.655 milisegundos
```

Abajo están algunos tiempos de ejecución típicos. El tiempo requerido para ejecutar el lazo FOR-NEXT de medición de tiempo en las líneas 5000 a 5090 mostrados arriba, fue restado de los valores abajo. En todos los casos fue ejecutada una instrucción RETURN en la línea 110.

Tiempos de ejecución típicos

Operación	**Tiempo** [ms]
100 A = 16/2	3.17
100 PRINT "STRING de prueba",: REM en 1200 baud	90.1
100 PRINT "STRING de prueba",: REM en 2400 baud	44.4
100 PRINT "STRING de prueba",: REM en 4800 baud	21.5
100 PRINT "STRING de prueba",: REM en 9600 baud	10.7
100 PRINT "STRING de prueba",: REM en 19.200 baud	5.65
100 PRINT "STRING de prueba",: Rem en 38.400 baud	3.25
100 A = 2**7	18.48
100 BITS=A: REM descifra un registro de E/S de 8 Bits	.57



Sugerencias para hacer más rápidos sus programas

Durante la ejecución de programa, cuando BASIC ampliado de FACTS ENGINEERING encuentra una nueva línea de referencia tal como "GOSUB 6000", explora el programa entero que comienza en el número de línea más bajo. Por lo tanto, las líneas referidas con frecuencia se deben poner tan cerca del inicio del programa como sea posible.

Por ejemplo, un GOSUB con una declaración RETURN en el final de un programa largo podría necesitar 8 milisegundos para ejecutarse. Un GOSUB con una declaración RETURN más cerca del comienzo del programa en el mismo programa (1 RETURN) podría requerir solamente 1,5 milisegundos para ejecutarse.

Las variables que se encuentran primero durante la ejecución de un programa BASIC se asignan en el comienzo de la tabla variable. Definiendo "Z" como la décima variable en un programa causó la declaración "IF Z=2 THEN END" para ejecutarse en 1,30 milisegundos.

Definiendo "Z" primero en el programa hace la misma declaración ejecutarse en 1,11 milisegundos (se gana el + 15%).

Un nombre de ocho caracteres puede requerir el 20% más tiempo de interpretar que una variable de un sólo carácter. Asimismo, una variable con dimensión puede requerir 35% más tiempo de procesar. El tiempo para una variable de un carácter es típicamente menos de 1 milisegundo.

Un método más rápido de solucionar el problema proporcionará a menudo el aumento más significativo de velocidad. Por ejemplo, se podrían usar cálculos exponenciales para descifrar el estado de los puntos de entradas y salidas del PLC. Sin embargo, la declaración BIT realizará típicamente la misma tarea 10 veces más rápidamente.

LISTA DE DECLARACIONES Y OPERADORES



En este apéndice:

Comandos

AUTOSTART	Selecciona el modo de operación de los módulos después de un reset
DELPRM	Borra un programa almacenado
CONT	Reanuda la ejecución de programa
EDIT	Mueve un programa archivado al PROGRAMA 0 para modificar (EDIT)
LIST	Muestra el programa corrientemente seleccionado
NEW	Borra el PROGRAMA 0 y las variables
NULL	Agrega caracteres nulos después de que cada retorno de carro
PROGRAM	Selecciona un programa archivado
RESET	Ejecuta un reset del software
RUN	Borra las tablas variables y ejecuta el programa seleccionado
SAVE	Almacena el programa seleccionado en el archivo del programa

Control de flujo

BREAK S	Habilita y deshabilita la parade del programa con <*CTRL-C*>
CLEAR	Reset los stacks de control y stacks de argumento
DO UNTIL	Ejecuta cálculos en un lazo hasta que la prueba en la parte inferior del lazo sea VERDADERA
DO WHILE	Ejecuta cálculos en un lazo mientras la prueba en la parte inferior del lazo sea VERDADERA
END	Para la ejecución de un programa
GO PRM	Comienza la ejecución de un programa especificado
GOSUB	Ejecuta una subrutina
GOTO	Transfiere la ejecución a la línea de programa especificada
IF	Ejecución condicional de declaraciones
LOCKOUT	Fuerza la ejecución de un programa
ON-GOSUB	Llama una subrutina que comienza en una de varios númerosde línea posibles
ON-GOTO	Salta a uno de variosnúmeros de línea posibles
ONERR	Especifica la línea de programa para ir si ocurre un error aritmético
RETURN	Marca el final de una subrutina
STOP	Para la ejecución de un programa

Entradas/salidas

BIT(S)	Descifra las entradas del PLC y codifica las salidas del PLC
CR	Utilizado en la declaración PRINT para hacer salir un **retorno de carro**
DATA	Especifica las expresiones para las declaraciones READ
INLEN	Vuelve el número de caracteres que esperan en un buffer (almacenador intermediario) de entrada o despeja especificado
INPLEN	Devuelve la cantidad de caracteres INPUT
INPUT	Carga variables con datos del puerto 1, 2 o 3.
PH0./PH1.	Imprime números hexadecimales de 2 y 4 dígitos
PICK	Opera en números enteros de 16 bits en un byte, nibble o bit
PRINT	Transmite datos para fuera del puerto serial especificado
POP	Recupera un valor del "stack"
PUSH	Pone un valor en el "stack"
READ	Asigna a declaración DATA valores constantes a las variables
RESTORE	Permite que valores constantes en una declaración DATA sean leídos otra vez
SETINPUT	Configura la declaración INPUT (INPUT$, LINE INPUT # y más)
SETPORT	Configura un puerto de comunicaciones.
SPC	Usado en la declaración PRINT para colocar una cantidad de espacios
TAB	Usado en la declaración PRINT para especificar la posición donde imprimir
USING	Coloca en formatos predefijnidos valores impresos y strings
@(y, x)	Colocación de cursor usando una secuencia de escape ANSI

Interrupciones

CLEAR I	Deshabilita las interrupciones del programa
IDLE	Suspende la ejecución de programa hasta la interrupción
ONPORT	Especifica el número de línea inicial para la dirección del evento del puerto serial
ONTIME	Interrupción por tiempo del flujo de programa normal
RETI	Marca el final de una subrutina que maneja una interrupción
TIME	Configura y recupera el valor del temporizador de software controlado por CLOCK

Operadores matemáticos

ABS	Devuelve el valor absoluto de una expresión
INT	Devuelve la porción del número entero de una expresión
SGN	Devuelve +1 si una expresión es mayor de cero, cero si una expresión es igual a cero, y -1 si una expresión es menos de cero.

SQR	Calcula la raíz cuadrada de una expresión
LOG	Calcula el logaritmo natural de una expresión
EXP	Eleva el número "e" (2.7182818....) a la potencia de una expresión
SIN	Calcula el seno de una expresión
COS	Calcula el coseno de una expresión
TAN	Calcula la tangente de una expresión
ATN	Calcula el arcotangente de una expresión
RND	Calcula un número pseudo-aleatorio entre 0 y 1 inclusive

Gerencia de memoria

CLEAR	Borra la memoria de una variable
COPY	Copia un bloque de memoria de ABM
DIM	Asigna la memoria para los arreglos numéricos
LOF	Devuelve el tamaño del programa actualmente seleccionado
STRING	Asigna la memoria para el almacenaje de string

Misceláneo

DELAY	Insiere una pausa
DTR	Controla la salida de la línea de "handshake" de hardware
ERRCHK	Genera la suma de comprobación, LRC, o los caracteres de verificación de error CRC-16 en una STRING o un bloque de memoria
LET	Asigna un valor a una variable
REM	Permite colocar comentarios no ejecutables en el programa
SYSTEM	Lee y configiura varios parámetros del sistema
TRACE	Ejecución de programa TRACE

Operadores de string

ASC	Cambia o devuelve el código ASCII de un carácter en una STRING
CHR$	Convierte un código ASCII en una sola cadena de caracteres
DATE$	Configura y recupera la fecha del reloj del calendario
INSTR	Busca en una STRING una STRING patrón
LCASE$	Devuelve una STRING que consiste en caracteres minúsculos solamente
LEFT$	Devuelve una cadena de caracteres de n que comienza con el primer carácter
LEN	Devuelve el número de caracteres en un carácter de string
MID$	Devuelve una cadena de caracteres de m que comienza con el enésimo carácter

RIGHT$	Devuelve una cadena de n caracteres que comienza con el último carácter
STR$	Devuelve el equivalente de string de una expresión matemática
TIME$	Configura y recupera la hora, minutos y segundos del reloj del calendario
UCASE$	Devuelve una STRING que consiste en caracteres mayúsculos solamente
VAL	Devuelve el equivalente numérico de una expresión de string
OCTHEX$	Convierte un número octal en su equivalente string hexadecimal ASCII

Avanzado

BYTE	Lee o escribe un valor de byte en la memoria variable de almacenaje
CALL	Invoca un subrutina de Assembler o de lenguaje de máquina
CBY	Lee el contenido de la dirección de memoria en la memoria del almacenaje de programa
COMERR	Bit de error CRC-16
DBY	Escribe a posiciones de memoria especiales (memoria interna de CPU)
LOAD @	Recupera un número de coma flotante de seis bytes de memoria
MTOP	Limita la memoria disponible para el intérprete BASIC
STORE @	Almacena un número de coma flotante de seis bytes en la dirección de memoria especificada
WORD	Lee en o escribe a una posición de memoria específica de dos bytes

INDICE

A

B

C

D